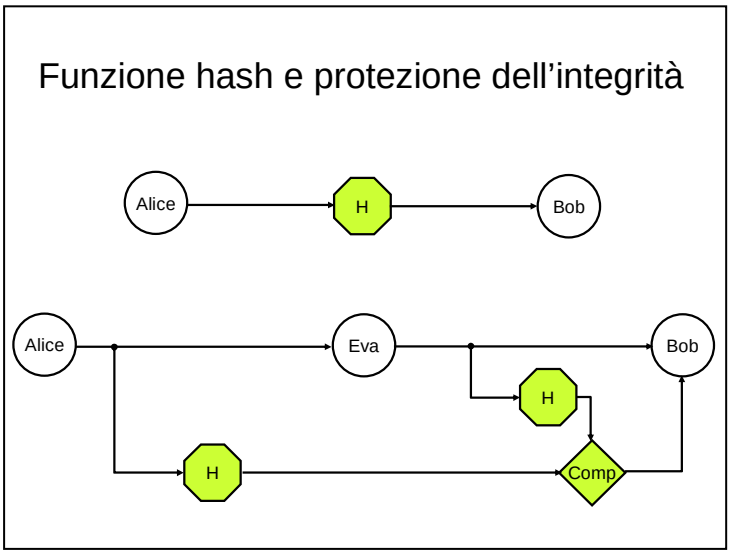
2. Esercitazioni consigliate

2.1 La funzione hash e l'accertamento dell'integrità di un messaggio

Obiettivo formativo – Prendere confidenza con la primitiva crittografica Hash e con il blocco Test. Costruire un sistema che consenta di verificare l'integrità di un messaggio. Capire la codifica base64 di un file binario.
Riferimenti: Capitoli 1, 2 e 6

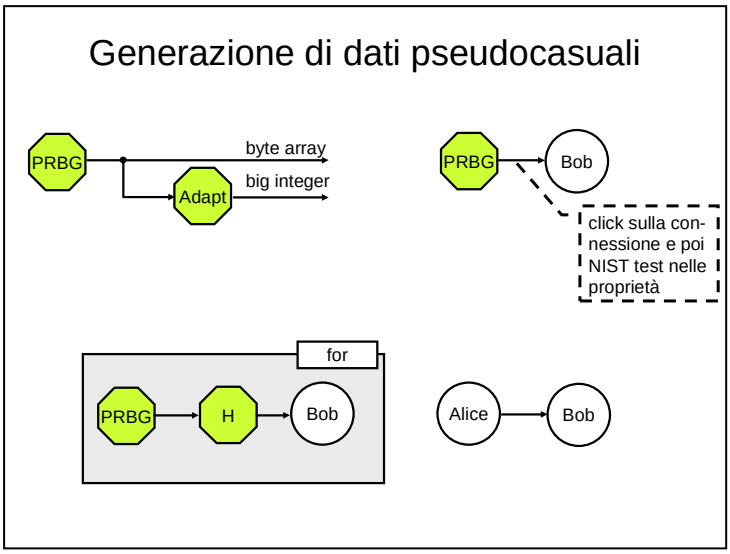


Esperimenti:

1. Dopo aver generato con Alice un messaggio di circa 1000 bit, istanziare, configurare ed eseguire un blocco Hash che lo riceva in ingresso.
2. Esaminare nella scheda Codice come si modifica la codifica in Java della funzione Hash dopo le tre fasi. Individuare la classe, i metodi ed i parametri impiegati nel codice (per semplicità in S-vLab si può qui usare solo il provider BC).
3. Calcolare l'impronta del messaggio con gli algoritmi MD5, SHA-1 e Tiger. Prendere atto dei valori (scheda In/Out) e dei tempi di esecuzione (scheda Codice). Ripetere la prova apportando lievi modifiche al messaggio.
4. Collegare Bob a valle di Hash, configurarlo con destinazione "stringa" e codifica "base64". Notare che la stringa generata da MD5 termina con due =, quella generata da SHA-1 con un =, mentre quella generara da Tiger non ha alcun = alla fine. Documentarsi su questa codifica (ad es. v. pag. 117) e giustificare le precedenti osservazioni.
5. Individuare l'algoritmo di hash che ha il più basso tempo di esecuzione per bit d'impronta.
6. Dimostrare con un esperimento che un algoritmo di hash impiega il principio della compressione iterata.
7. Modellare un sistema che consenta a Bob di verificare l'integrità dei messaggi che riceve da Alice. Esaminare la scheda Codice del blocco Comp.
8. Esaminare su Bob la risposta del blocco Comp in assenza ed in presenza di attacchi di Eva.

2.2 Il generatore di dati pseudocasuali

Obiettivo formativo – Prendere confidenza con la primitiva crittografica PRNG, con il blocco per la conversione di tipo e con la BOX-for. Verificare il modello dell'oracolo casuale.
Riferimenti: Capitolo 2

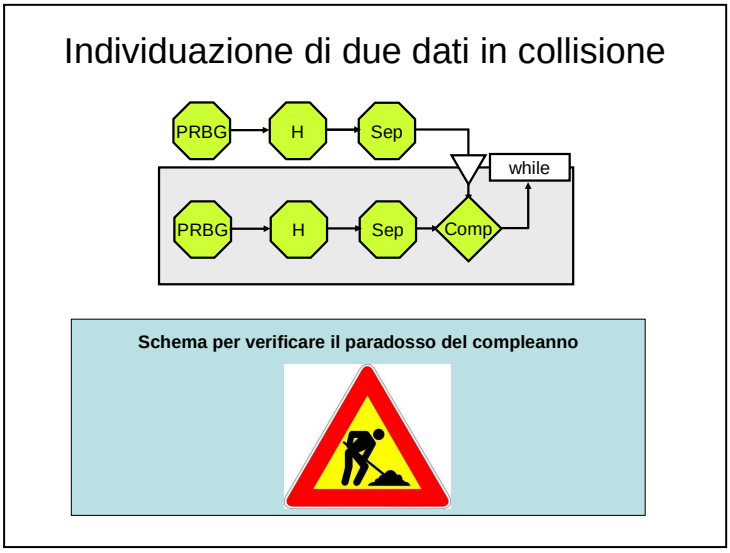


Esperimenti:

1. Istanziare un PRBG. Generare array di 160, 320 e 640 bit. Prendere atto dei tempi di esecuzione e giustificarli.
2. Esaminare la scheda Codice del PRNG.
3. Tramite l'esperto, esaminare su Javadoc la classe Secure-Random ed i suoi metodi.
4. Configurare Adapt per passare da array a big integer.
5. Esaminare la casualità di un array di 20000 bit generato dal PRBG eseguendo i primi 4 test del NIST; a tal fine attualmente occorre collegare Bob all'uscita del PRBG e fare le azioni indicate in figura.
6. Istanziare una BOX-for e porre al suo interno, in serie, un PRBG, un Hash e Bob. Configurare PRBG per dati di 512 bit, Hash per SHA-1 e Bob per destination file con append; configurare la BOX per 125 iterazioni. Istanziare e collegare Alice e Bob fuori dalla BOX, configurare Alice per inviare il file generato precedentemente da Bob e sottoporlo ai test del NIST.
7. Esaminare la scheda Codice della BOX-for

2.3 Generazione di due dati con la stessa impronta

Obiettivo formativo – Prendere atto della possibilità di individuare due dati in collisione. Imparare ad usare la BOX-while.
Riferimenti: Capitolo 2

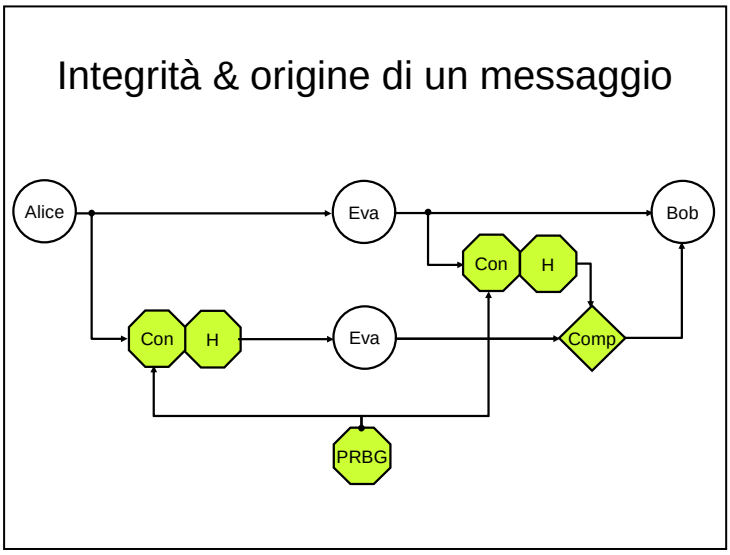


Esperimenti:

1. Disporre in cascata un PRBG, un Hash ed un Separatore. Configurare il Separatore in modo da disporre di impronte di 8 bit.
2. Istanziare una BOX-while ed istanziare al suo interno la cascata PRBG-Hash-Sep-Comp. Inviare all'altro ingresso di Comp l'impronta calcolata al di fuori della BOX e l'uscita di Comp al While. Eseguire prima la parte fuori dalla BOX e poi la BOX. Esaminare la scheda Codice della BOX. Prendere nota del n°di iterazioni
3. Ripetere alcune volte l'esperimento e giustificare i valori del n° di iterazioni che si sono verificati.
4. La verifica del paradosso del compleanno, a causa di componenti che devono ancora essere realizzati, verrà fatta in una prossima esercitazione.

2.4 L'accertamento dell'origine e dell'integrità di un messaggio

Obiettivo formativo – Prendere atto dell'operazione di concatenazione. Costruire un sistema che consenta di accertare l'integrità e l'origine di un messaggio utilizzando un segreto condiviso dagli utenti.
Riferimenti: Capitoli 1 e 2



Esperimenti:

1. Modellare un sistema che consenta a Bob di verificare integrità e origine dei messaggi ricevuti.
2. Generare con un PRBG il segreto che Alice e Bob devono condividere, attribuendogli una dimensione adeguata.
3. Notare che i due blocchi di concatenazione non hanno parametri. Prendere atto sulla scheda Codice della tecnica con cui viene costruito l'array d'uscita. Scegliere un algoritmo di hash.
4. Eseguire passo-passo i componenti del modello. Esaminare cosa riceve Bob in assenza ed in presenza di attacchi sul messaggio e sull'impronta.
5. Tramite l'esperto, attivare il link contenuto nella cartella "MD5" e leggere il documento sulle collisioni che sono state trovate. Prelevare dal sito del corso i file binari MD5collisionN1.bin e MD5collisionN2.bin e copiarli nella cartella file del vostro progetto.
6. Assegnare ad Alice e ad Eva rispettivamente il compito di generare sulle loro uscite i due file binari (HEX come codifica!), configurare Hash con MD5 e lasciando transitare sul canale, senza modifiche, l'impronta calcolata da Alice. Controllare l'effetto di questo attacco e giustificarlo.
7. Ripetere la prova facendo calcolare H(H(m||s)||m) ad Alice ed a Bob.

REPORT N.2

2.1 La funzione hash e l'accertamento dell'integrità di un messaggio

Immagine dello schema studiato	**Osservazioni:**

2.2 Il generatore di dati pseudocasuali

Immagine dello schema studiato	**Osservazioni:**

2.3 Generazione di due dati con la stessa impronta

Immagine dello schema studiato	**Osservazioni:**

2.4 L’accertamento dell’origine e dell’integrità di un messaggio

Immagine dello schema studiato	**Osservazioni:**