International Institute of Information Technology, Hyderabad.
Principles of Information Security

Evaluation III

April 3, 2020

Due: **April 10, 2020**.

Instructions : Two Evaluation sheets will be released every week (on Tuesdays and Fridays). Each evaluation sheet consists of three categories of questions, namely: **[P]** stands for *programming* assignment, **[Q]** stands for *question* with written solution to be submitted and **[R]** stands for *research* problem. You need to submit the source-code for **[P]** along with a screen-recorded video that demonstrates its execution and for **[Q]** you may submit a pdf-file solution, all by the due-date. The research problems are *optional*, and anyone who solves any *one* of the **[R]** problems among *all* evaluation sheets will directly be awarded an **A** grade.

[Q] Recall that a *pointer* to data X stores the *address* of X, denoted by &X. Similarly, let a *hash-pointer* to data X be the address of X along with the cryptographic hash (say, H) of X, denoted $\langle \&X, H(X) \rangle$. Further, let a *hash&sign-pointer* to data X be the address of X, along with the cryptographic hash (say, H) of X that is digitally signed by the owner of X, denoted $\langle \&X, H(X), \sigma \rangle$. Let $\mathbb{D}$ be your favourite data structure among `stacks, queues, lists, trees, forests, graphs, priority-queues` along with their variants (like Binomial heaps, skip-lists, red-black-tress and so on). Consider a **pointer-based** implementation of $\mathbb{D}$. It is straight-forward to visualize the corresponding **hash-pointer-based** implementation of $\mathbb{D}$, wherein every pointer &X is replaced by $\langle \&X, H(X) \rangle$. What do you think are the advantages of the hash-pointer-based implementation of $\mathbb{D}$ over the pointer-based implementation of $\mathbb{D}$? Specifically, can you think of one application/problem/setting/protocol, say $\mathcal{A}_{hash}$, wherein a hash-pointer-based implementation of $\mathbb{D}$ is more suitable? Analogously, list the advantages of the hash&sign-pointer-based implementation of $\mathbb{D}$ and give an application $\mathcal{A}_{sign}$ where it is (more) suitable. Justify your answers.

[P] Implement (in any popular programming language of your choice) all the three versions of $\mathbb{D}$, namely, pointer-based, hash-pointer-based and hash&sign-pointer-based, using your own collision resistant hash function and digital signature scheme (implemented by you in *Evaluation I*).

ALL THE BEST

[R] **Computing $[m,k]$-(s,t)-Weak-Connectivity:** A digraph $G = (V, A)$ is said to be **k-(s,t)-connected** if, on the deletion of any k nodes (other than s and t) from G, there still exists a path from s to t. There are several famous polynomial-time algorithms to compute the maximum **k** such that G is **k-(s,t)-connected** (for example, many are based on algorithms for MAX-FLOW). Let V^* denote set of vertices that have a path to t, i.e. $V^* = \{u | \exists \text{ path from } u \text{ to } t \text{ in } G\}$. The digraph $G = (V, A)$ is said to be **k-(s,t)-weak-connected** if, the underlying undirected graph G_u (that is, the graph obtained by replacing every arc in G by an undirected edge) when induced on V^*, denoted $G_u[V^*]$ (that is, we delete all the vertices in $V \setminus V^*$ in G_u), is **k-(s,t)-connected**. The digraph $G = (V, A)$ is said to be **[m,k]-(s,t)-weak-connected** if, on the deletion of any m nodes from G, the remaining graph is **k-(s,t)-weak-connected**. The question is: *given a digraph G, and integers m and k, design an efficient algorithm to decide if G is* **[m,k]-(s,t)-weak-connected**.