Laboratory exercise (1 student)
One-time password authentication

Consider the following (naïve) entity authentication protocol for A who wants to prove his identity to the verifier B.

A chooses a password p_A with up to L_max decimal digits and a one-way hash function $h : \{0, 1, \dots, 9\}^{L_\mathrm{max}} \to \{0, 1, \dots, 9\}^{L_\mathrm{max}}$. Let x_1 be the hash, and x_N be the N-fold hash of the (possibly zero-padded) password p_A, that is

$$x_1 = h([p_A, 0, \dots, 0]) \quad , \quad x_N = \underbrace{h \circ h \circ \dots h}_{N}([p_A, 0, \dots, 0]) \ .$$

In the protocol setup phase, A securely delivers x_N to B. Then, in the n-th run of the protocol, $n = 1, \dots, N-1$

$\boxed{1}$ A → B : $u_1 = [\mathrm{id_A}]$

$\boxed{2}$ B : $r_n \sim \mathcal{U}(\mathcal{R})$
B → A : $u_2 = [n, r_n]$

$\boxed{3}$ A : $x_n = h(x_{n-1})$
A : $u_3 = [x_n, r_n]$
A → B : u_3

$\boxed{4}$ B : checks whether $\underbrace{h \circ h \circ \dots h}_{N-n}(x_n) = x_N$ and the received r_n is consistent with the transmitted one, and if so, B accepts A.

Your assignment is to:

1) implement the above protocol in your favourite language (Matlab/Octave, C, Java, or Python are all fine), and check its correctness;
2) identifiy its vulnerabilities and devise an attack that exploits them, under reasonable assumptions;
3) implement the attack and evaluate its success probability in dependence of the protocol parameters;
4) suggest improvements to the protocol and implement them.

Provide a description of your solution, with justification of your choices, the code for your implementations, and adequate discussion of the results.

General notes: Implementing your protocol may require you to use several cryptographic primitives, such as symmetric or asymmetric encryption/decryption, signing/verification, one-way functions, cryptographic hash functions, key generation, etc. Feel free to use any reasonable (and correct) implementation of such primitives you find for the language you've chosen, or even simplified models such as a (pseudo-)random oracle. The vulnerability of the protocols should not depend on a poor implementation of such primitives. Also, unlike for cryptographic keys, you should model the random choice of a password as (strongly) non uniform.