Laboratory exercise: one-time password authentication

Aim, scope and working hypothesis:

The given naïve one-time password authentication protocol ($\mathrm{M_{naïve}}$) and its improved version ($\mathrm{M_{impr}}$) are entity authentication protocols aiming at assessing the identity of an entity before a communication session. If we consider an attacker able to interrupt or delay our communications, or even worse to act as man-in-the-middle, the only thing we can successfully try to prevent with entity authentication services is the arbitrary initiation of a new session leading to a successful masquerade. If we tried to prevent a more general class of attacks, we would be forced to make constraining assumptions on the attacker and its capabilities, like imposing him to perform only passive attacks and execute the authentication protocol. On the contrary we will only require the secrecy of the chosen password, the effective one-wayness of the hash function and the impossibility for the attacker to force A to start an authentication session.

Correctness of the naïve protocol:

The correctness of the protocol in the sense that a legitimate entity's identity is correctly verified, is straightforward to check. Indeed, in step 4 we have B checking if $x_\mathrm{N} = \underset{\text{N-n times}}{h \circ \ldots \circ h(x_\mathrm{n})}$ but

$$\underset{\text{N-n times}}{h \circ \ldots \circ h(x_\mathrm{n})} = \underset{\text{N-n times}}{h \circ \ldots \circ h(h(x_{n-1}))} = \underset{(\mathrm{N}-\mathrm{n}+1)\text{ times}}{h \circ \ldots \circ h(x_{n-1})} = \ldots = \underset{(\mathrm{N}-1)\text{ times}}{h \circ \ldots \circ h(x_1)} = x_\mathrm{N}$$

by construction so the check will give a positive result whenever the protocol is followed by a legitimate prover and without malicious interferences.

Vulnerabilities and possible attacks:

The security level against an ignorant attack I where an attacker E try to guess a password, having a correct estimate of its distribution is given by the min-entropy of the password probability mass function. Indeed, if we consider the hash function as bijective, the optimal strategy for an ignorant attacker is to use as first guess the most probable password, and then hash it the appropriate number of time. Then the success of the masquerade only depends on the coincidence of the guessed password to the actually used one (p_A) and its probability is given by:

$$\mathrm{Prob}(\mathrm{S_I}|\mathrm{I},\mathrm{M_{naïve}}) = \mathrm{Prob}(\boldsymbol{p_A} = \underset{\mathrm{k}\in\{0,1,\ldots,9\}^{\mathrm{L_{max}}}}{\operatorname{argmax}} \mathrm{Prob}(\boldsymbol{p_A} = \mathrm{k})) = \underset{\mathrm{k}\in\{0,1,\ldots,9\}^{\mathrm{L_{max}}}}{\max} \mathrm{Prob}(\boldsymbol{p_A} = \mathrm{k}).$$

The main vulnerability of this protocol lies in the fact that if an attacker E is able to eavesdrop the channel during some ith run of the protocol between A and B then he will be able to successfully authenticate himself as A. Indeed suppose that E knows $x_i = h \circ \ldots \circ h([p_A, 0, \ldots, 0])$, where the hashing has been applied i times, then he will be able to initiate a new authentication session with B by sending a message containing $\mathrm{id_A}$ and then, after having received the challenge $[j, r_j]$, to compute $x_j = h \circ \ldots \circ h(x_i)$, where the hashing must be applied $\mathrm{j}-\mathrm{i}$ more times, and send it back alongside r_j to B. Then B will validate the authentication of E as A because the received answer to the challenge is the correct one with probability 1. Following Kerckhoffs' principle we cannot rely on the secrecy of the used hash function, as in this context it is not parametrized on a secret key which would require a renewal policy and confidential sharing.

Proposed improvements:

To solve this problem only a little change is needed: we have to reveal, during the legitimate runs, the hash functions in decreasing order of hashing, that is, starting from the key hashed N times (which has been secretly shared) A transmits at each run the key hashed one time less than the previous one. In this way an attacker will not be able to compute the successive answers to the challenges from the previous ones because this would require him to invert the hashing function, but this is not feasible in reasonable time because the hash function is one-way, so computational security is guaranteed. A can compute the hashes each times it needs one or store all of them while computing step by step x_{N} at the beginning of the protocol. Only step 3 and 4 of the protocol must be modified in the following way, after having defined $x_{\mathrm{i}} \triangleq \underset{\text{i-1 times}}{h \circ \ldots \circ h(x_1)}$, we at the *n*^th run :

3 A: $\rightarrow \mathrm{u}_3 = [x_{\text{N-n}}, \mathrm{r}_{\mathrm{n}}]$
A $\rightarrow$ B : u_3

4 B: checks whether $\underset{\text{n times}}{h \circ \ldots \circ h(x_{\text{N-n}})} = x_{\mathrm{N}}$ and the received r_{n} is consistent with the transmitted one, and if so, B accepts A

Correctness and security of the implemented solution:

As previously shown, to check the validity of the received hash the verifier will now have to apply the hashing function *n* times in the *n*th run of the protocol, so correctness is guaranteed:

$$\underset{\text{n times}}{h \circ \ldots \circ h(x_{\text{N-n}})} = \underset{\text{n + (N-n-1) times}}{h \circ \ldots \circ h(x_1)} = \underset{\text{(N} - 1\text{) times}}{h \circ \ldots \circ h(x_1)} = x_{\mathrm{N}} \text{ iff } x_1 = h([p_A, 0, \ldots, 0]) \ .$$

Now that we have avoided the possibility of performing the attack resulting from the vulnerabilities of the naïve protocol, let's see what options are left to an attacker willing to perform a masquerade. In order to obtain numerical results we can model the hash function chain of values ideally as an arbitrary permutation of the elements of the set $\{0,1,\ldots,9\}^{\mathrm{L_{max}}}$. Consider the hash function composition to be denoted:

$$h^{\mathrm{k}}(\mathrm{u}) \triangleq \underset{\text{k times}}{h \circ \ldots \circ h(\mathrm{u})},$$

in such a way that for any starting point u, $[h^1(\mathrm{u}), h^2(\mathrm{u}), \ldots, h^{10^{\mathrm{Lmax}+1}-1}(\mathrm{u})]$ is an unknown permutation of the elements of the set $\{0,1,\ldots,9\}^{\mathrm{L_{max}}}$. Now suppose that the N-1 runs of the protocol have been observed by the attacker, he can consider that the password isn't any of the N-1 hashes he has seen. In the case of uniformly distributed password, a random guess between the remaining elements is the optimal solution. This will provide a password which must be hashed in average $\frac{(10^{Lmax+1}-1)-N-1}{2}$ times to obtain the desired value of x_2 . Given that we will choose N many orders of magnitudes lower than 10^{Lmax+1} , we can affirm that obtaining a valid hash in this way can be considered computationally unfeasible in a reasonable time.

Further improvements for other attacks:

Authentication systems must typically be able to verify the identity of a large number of users. In the naive protocol x_{N} is said to be delivered securely in the setup phase. What interest us the most is not the secrecy of the transmission of x_{N} but the fact that its integrity is preserved in the transmission. Anyway let's suppose that an attacker could be able to eavesdrop the transmissions of x_{N} and the corresponding identifier for multiple users, or to obtain them by accessing to the storage of the verifier. Then the attacker could try to compute the (N times) hashed versions of the most likely passwords and check for any correspondence with the ones previously obtained. The probability to guess a password actually used is thus multiplied by the number of observed x_{N}. In this optic salting could be used to increase the temporal complexity of such an attack (or more elaborated ones like rainbow tables).

Additional notes:

I chose to implement the algorithms in Python, the files that need to be ran are "Naive_protocol.py" and "Improved_protocol.py. They will ask the password as input in the console. I also joined Jupyter Notebooks for a more easy visualisation.