Home Assignment 1

Advanced Web Security

2018

B-assignments

For grade 3, complete the three B-assignments below and solve them in groups of ≤ 2 students.

B-1 Implement the Luhn algorithm in your favorite language. You will be given a list of card numbers with one digit censored with an "X". Use your implementation to find the censored digit in order for the card number to be valid. You will be given a text file, consisting of 100 card numbers, one per line. The answer is the concatenation of all censored digits, in order according to the list.

Example: Let the input list be

```
12774212857X4109
586604X108627571
7473X86953606632
4026467X45830632
20X3092648604969
```

The answer is given by the numeric string "54963".
Assessment:

- Upload your code to Urkund, jonathan.sonnerup.lu@analys.urkund.se.
- There will be one Moodle questions following the problem statement above. Both students must finish the Moodle quiz. There will be a test quiz on Moodle, where you can try your implementation as many times as you like. The test quiz will not be graded.

B-2 Read sections 4 and 4.1 in the MicroMint paper[1]. Write a program that simulates the time needed to generate MicroMint coins. Model the process as a balls-and-bins problem. Your program should take three parameters; u, k and c. The parameter u is the number of bits used for identifying the bin, so there are $b = 2^u$ bins. Parameter k is the number of collisions (balls in the same bin) needed to make a coin, so that fewer than k balls in a bin make no coins, and k or more balls in a bin make precisely one coin. All bins are empty at first. At each iteration, throw a ball into a randomly selected bin. How many iterations (how many balls thrown) does it take before you generate $c = 1,100$ and 10,000 coins, respectively? You will be given u, k and c.
Note: With respect to the MicroMint paper, use $t = 0$.
The above procedure describes one simulation. You will need to run several such simulations in order to compute **the averages** reliably. Construct 99.9% confidence intervals for the three mean values. So, you may ask, for how long do I need to run these simulations? You will be given appropriate widths for your confidence intervals. With n statistical observations $x_1, x_2, \ldots, x_n$, a 99.9% confidence interval for the mean is given by

$$\left(\bar{x} \pm \lambda \cdot \frac{s}{\sqrt{n}}\right),$$

[1] R. L. Rivest and A. Shamir - PayWord and MicroMint: Two simple micropayment schemes

where $\bar{x}$ and s are your sample-estimated average and standard deviation, respectively. You may use $\lambda = 3.66$. Below we present the minimum and maximum values for u and k that you should be able to handle. Make sure you can handle all integer values between these extremes. **Note**: A simulation should take no longer than 30 minutes.

Example: In the tables below you are given two examples of simulations results. In the first example, the simulation is run with the parameters $u = 16, k = 2, c = 1$. The mean is calculated to be 322. Note that this value is just an estimate of the *true* mean value. If you create a confidence interval of width 22 (in this case) for the mean, then you will reliably assess the true mean value, which will be enough to pass this assignment.

Table 1: Simulation with $u = 16, k = 2$.

# coins (c)	mean ($\bar{x}$)	c.i. width
1	322	22
100	3685	24
10k	45270	22

Table 2: Simulation with $u = 20, k = 7$.

# coins (c)	mean ($\bar{x}$)	c.i. width
1	493981	79671
100	1069997	15616
10k	2420113	4783

Assessment:

- Upload your code to Urkund, jonathan.sonnerup.lu@analys.urkund.se.
- There will be one Moodle question following the problem statement above. Both students must finish the Moodle quiz. There will be a test quiz on Moodle, where you can try your implementation as many times as you like. The test quiz will not be graded.

B-3 Lightweight Bitcoin clients, so-called Simplified Payment Verification (SPV) nodes, that only store the block headers may wish to verify that a transaction is included in the blockchain. To do this, the SPV node queries a full node for the Merkle path of a certain transaction. The path is used to compute the Merkle root, which then can be compared to the Merkle root stored in block header. In this problem, you will take on the role of both an SPV node that verifies the transaction using the Merkle path, and also a full node providing the Merkle path for a given transaction.

Part 1, SPV node.
Implement a program that takes a file with a leaf node L, given as a hexadecimal string, and a Merkle path as input, with each node in the path on a separate line. The $2t$-byte strings should be interpreted as t-byte byte arrays. The Merkle path nodes are given in the order of highest depth first, i.e., the leaf node sibling. By convention, the root node is at depth 0 and the leaves are at depth $\lceil \log_2 n \rceil$, where n is the number of leaves. Each string representing nodes in the Merkle path is preceded by the letter 'L' or 'R', indicating if the sibling node in the path is a *Left* or *Right* node. The program should output the Merkle root as a hexadecimal string. Use SHA-1 as hash function throughout the tree and merge nodes using concatenation of SHA-1 results (byte arrays).

Example Using a file of the format

```
2354cf006ef4eeefeddf29b9e68d5cb1918ed589
R69968f8d734080390646bd0f3afff78baadebd2b
Led64e17870e63f55b71542f0818ff7639b1f9985
L7a6ba60c80a893b7a02999b6415c6ec67d5883b4
L64b64c7760e5559aefe701790ee0564af6458cb4
L53f1eab7ccd09600908bc49044669cd8fc996171
Rc5684eb22d8745a777037c19ff3eff85be800334
L058e2c0d7d103a7b45b2a4408ac3389eb10048fe
```

the node `2354cf006ef4eeefeddf29b9e68d5cb1918ed589` is concatenated with the first node in the path, `69968f8d734080390646bd0f3afff78baadebd2b`. The SHA-1 hash if this concatenation is then appended to `ed64e17870e63f55b71542f0818ff7639b1f9985` and so on. The resulting Merkle root is `6f51120bc17e224de27d3d27b32f05d0a5ffb376`.

Part 2, Full node.
Implement a program that takes an integer index i, another integer index j, and a set of leaves, $(\ell(0), \ell(1), \ell(2), \ldots, \ell(n-1))$, in a Merkle tree. The leaves are given as hexadecimal strings as input, but should be interpreted as byte arrays. The input is summarized in a file, starting with the integer index i, the integer index j, and then followed by the leaves, one input on each line. Your program should provide:

1. The Merkle path for leaf $\ell(i)$, starting with the sibling of $\ell(i)$.
2. The Merkle path node at a given depth j (this will be used in the assessment).
3. The resulting Merkle root (this will be used in the assessment).

Specifically, the output of your program should provide a concatenation of items 2 and 3 above. Make sure you support an arbitrary number of leaves. The format of the nodes should be hexadecimal strings. Each string representing nodes in the Merkle path must be preceded by the letter 'L' or 'R', indicating if the sibling node in the path is a *Left* or *Right* node. While building the tree, starting with the leaves, if the set of nodes on depth k is odd, simply append a copy of the last node to get an even number of nodes.

Example Using a file of the format

```
9
2
fcfdd2a12d8b0b75c2edb47d691470dd0178c566
031e3057d40d25472cdb80e4952ed6738936aba8
7164148d695971de1b9f31be8e730299823b1def
64d866d14053a33e75744d966daa4f47b92018b9
a7e62ee4822cb29ec7004841d8f819965be169d9
c176f7a3dc7a3f59136b42163efa8f6350bb073c
c25053cedc71463287168687accf4e701a7ace0f
2af15e40605eb51d3a2e041f9799f3a070398eb1
053dfa25bf07b0d530c6bb9f803d60062f4e0e00
6c136b269efa9535f02732d68c80371625783cfc
d1705b195fb31e4e9c0b03570c677e59a19250b8
e5fb3dc4357332ec108e5d4d25991942e4ec9a95
```

will result in a Merkle tree with the leaves at depth 4. The Merkle path for the node with index 9, i.e., `6c136b269efa9535f02732d68c80371625783cfc`, is given by

```
L053dfa25bf07b0d530c6bb9f803d60062f4e0e00
R9befe43f5e8e4dce20616129c9fa1b782c080e73
R8d3f164890509c6510cc9bc975cb978f0b872fbb
Laa34d87b649005b70e9ea55390679e84c7627d1c
```

These nodes are on depth 4, 3, 2 and 1 respectively (starting at the top of the list). Thus, the Merkle path node at depth 2 is `8d3f164890509c6510cc9bc975cb978f0b872fbb` and this node is Right sibling. The Merkle root is given by `1781a6ea9a22f67e8a09cb54bbdc6d99d0efc081`. The concatenation of these is given by

```
R8d3f164890509c6510cc9bc975cb978f0b872fbb1781a6ea9a22f67e8a09cb54bbdc6d99d0efc081
```

Note that the R has been added to the answer.

Assessment:

- Upload your code to Urkund, jonathan.sonnerup.lu@analys.urkund.se.
- There will be three Moodle question following the problem statement above (one Merkle path node and two Merkle roots). Both students must finish the Moodle quiz. There will be a test quiz on Moodle, where you can try your implementation as many times as you like. The test quiz will not be graded.

C-Assignment

For grade 4, complete the C-assignment below and solve it in groups of ≤ 2 students.

C-1 Implement the coin withdrawal (the version with 2k quadruples) in the untraceable e-cash scheme given in the lecture notes.

- You may use a variant of RSA with easily manageable numbers. Note though that too small numbers might get you in trouble when you try to unblind if you are not careful.
- The data transfer can be simulated locally inside the program.
- The extended Euclidean algorithm can be used to find the inverse of 3 mod x. You are free to use existing code for this algorithm as long as you reference it properly.
- Choose sensible functions f and h.
- Be careful when choosing your r_i. Not all choices are suitable.

Assessment:

- Summarize your work in a short report, making it clear that the program works as intended.
- Upload your code to Urkund, jonathan.sonnerup.lu@analys.urkund.se. Only submit it once.
- Upload your report to Urkund, jonathan.sonnerup.lu@analys.urkund.se. Only submit it once.
- Upload your report to Moodle (it will be manually graded). BOTH students must do this.