EITN41 - Advanced Web Security

Assignment 3B.1

Oliver Elias Nilssen (ol0716ni)
5th of December 2020

Lund University
Sweden

1 Intro

In this short report we will discuss the binding and concealing property of a commitment scheme. We will illustrate the finding using a graph and talk a little bit about how an attacker would do to break the binding property or concealing property of this scheme.

2 Experiment

To simulate the breaking of binding and concealing properties, we used Python to program a "simulation" of someone trying to break the properties. To make the task simpler, and as instructed in the task description, the committer only commits X bit of his whole hash. Each commit is committed with a vote, either 1 or 0 and a random K which is concatenated to the vote. In this experiment we a value for K that started at 1 and ended at 2^{16} to put a limit on the amount of possible values. Which means we ended up with $2*2^{16}$ possible values, as we did it for both possibilities of votes. Each vote was then truncated down to X bits, starting at 1 and ending 30 bit length.

2.0.1 Binding property

The goal of this assignment was to check if we are able to find the same hashed value for a vote which is 1 by voting 0. We want to illustrator how the possibility of breaking the binding property is easier the smaller our committed hash is. The binding property is described as follows: "The Sender cannot "cheat" in the second phase and send a different key Kthat causes the commitment to open to a different message m." [1]

```
v = randint(0,2)
k = randint(2**16)
commit = (v + k)[-X:] #truncate

for i in range(2**16):
    new_commit = (opposite(v) + i)
    if new_commit == commit:
        return true
return false
```

Above we see a simplification of the algorithm used. The hash was of course hashed first and then truncated, which is not illustrated in this code. First a

compare commit was created, we create new commits based on the opposite voting value and all possible K's and only stop when we have a match/collision. We do this about 5 times for each value of X in the simulation. More simulations would be ideal, but since we are dealing with numbers going into the thousands, the simulation takes it's time running otherwise. We check for each value of X, and for all values of K for each X.

The result we get where not entirely surprising. We can see that when X is low, up until about 8-9 bits in length, the binding property is broken 100% of the time. e.g that means we have been able to find a match each iteration that says that a vote c*k=c'*k by checking all values for K (up until 2^{16})

Once we start committing longer values above 10 bit long, then the breaking of the binding property become harder and harder. This shows us how important it is for the length of the commit to be of a decent size so that it is harder to find matches. This also has to do with the birthday paradox. We know that if we have a group of 23 people, there is a 50% chance that there will be 2 people in the group that has a birthday on the same day. For a 99% match, we only need to reach 70 people. This is because we are comparing all the individuals against each other, say 23 people, that gives us a total of 253 comparisons, which is more than half the days in a year. This birthday paradox also applies to hashing. The expected collisions or matches to be made with hashing functions of N-bit hashes is $2^{N/2}$. This means that for us to have a low bit count on our hash we will much more easily find a match. That means we would need a decent length hashing function for there to be minimal collisions.[2]

In our tests we starts as said before by submitting truncated hashes starting at 1 bit up until 16 in length. Once we start reaching commits that are longer than 10 in length, we see a decrease in collisions. There will be a lot less hashes that are similar as every little bit appended to the end of our commit will add a bunch of new possibilities. For a commitment scheme to be successful, we would have to make sure that $c \neq c'$ for the scheme to be secure.

2.0.2 Concealing property

By concealing property we mean that when someone commits a message, the receiver should not be able to find out any information about the message.[3] So we want to make sure our committed info is concealed in a way that makes it more or less impossible to know what that commit contains in regards to say 1 or 0 for a 1 bit voting system.

In this test we did similar as before when it came to the simulations. We did a loop of X iterations (we used 30), and did the conceal and binding test in the same loop. And for each value of X, we did 5 tests on the concealing property to get an average. To get a more accurate result, we should have done more tests.

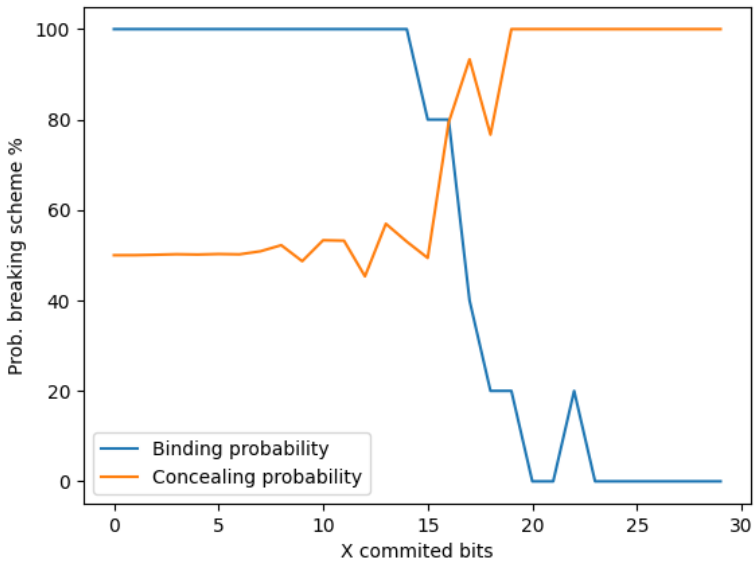
```
vote0, vote1 = 0, 1
v_org = randrange(0, 2)
k_org = randrange(0, 2**16)
org_commit = commit(v + k)[-X:]

commits_zeros = []
commits_ones = []
for k in range(2**16):
    cmt0 = commit(vote0 + k)[-X:]
    cmt1 = commit(vote1 + k)[-X:]
    if cmt0 == org_commit: commits_zeroes.append(cmt0)
    if cmt1 == org_commit: commits_ones.append(cmt1)

if (v == 1):
    return len(commits_ones)/(len(commits_zeros) + len(commits_ones))
else:
    return len(commits_zeros)/(len(commits_zeros) + len(commits_ones))
```

As can be seen in the algorithm above, we created two initial votes, one for 1 and one for 0. Then we did a test with 2^{16} values of K, and checked if the committed vote was the same as any of the randomly newly created commits (where the vote is the same, but the random-value is different). As described in the assignment text: "We define the probability of breaking the hiding property as the ratio between the number of possible x values for which the committed bit v can be uniquely determined and the total number of possible x values output by the hash function." [4]

On average to start with, the concealing property is about 50%, but once we increase the number of bits, it starts increasing and goes upwards to towards 100% after we reach about bit length of 10 for the commits.

3 Conclusion

Using the *matplotlib* in python, we managed to plot our values to give us a representation in a graph as how the two probabilities went. We applies a graph that on on lines measures length of X in bits, and the other side measures % of if the binding or concealing property was broken.



As can be seen in the graph, the results are described in the experiments above. We see that the longer the length of the commit, the harder it is to break the binding property, but it's easier to break the concealing property. We can't have a commitment scheme that is perfectly concealing and binding, but rather it's believed we can have one of the other.

The two attacks work in similar fashion, which is to analyze and try to break hashing functions. With binding we are trying to make sure that the user that sends a commit can't change their mind later by sending a different value. Which if you look at this experiment is possible if the hash is small enough. It makes it possible to find a hash which looks the same as a hash of a different value. This would also break the commitment scheme properties, as the idea is that "voters" shouldn't be able to change their mind after they committed something.

4 References

[1]: Luca Trevisan, Cryptography, [30.04.09],
$http://theory.stanford.edu/\ trevisan/cs276/lecture27.pdf$

[2]: Birthday Paradox,
$https://betterexplained.com/articles/understanding-the-birthday-paradox/$

[3]: Commitment Scheme,
$https://cryptography.fandom.com/wiki/Commitment_scheme$

[4]: Assignment info page,
$https://canvas.education.lu.se/courses/8222/quizzes/11159$