Handout 6 (Zero-Knowledge Proofs)

Zero-knowledge proofs (short ZKP) solve a paradoxical puzzle: How to convince somebody else that one knows a secret, without revealing what the secret actually is? This sounds like a problem the Mad Hatter from Alice in Wonderland would occupy himself with, but actually there some serious and not so serious applications of it. For example, if you solve crosswords with your friend, say Bob, you might want to convince him that you found a solution for one question, but of course you do not want to reveal the solution, as this might give Bob an advantage somewhere else in the crossword.

So how to convince Bob that you know the answer (or a secret)? One way would be to come up with the following protocol: Suppose the answer is *folio*. Then look up the definition of *folio* in a dictionary. Say you find:

> "an *individual* leaf of paper or parchment, either loose as one of a
> series or forming part of a bound volume, which is numbered on
> the recto or front side only."

Take the first non-small word[1] in this definition, in this case *individual*, and look up the definition of this word, say

> "a single *human* being as distinct from a group"

In this definition take the second non-small word, that is *human*, and again look up the definition of this word. This will yield

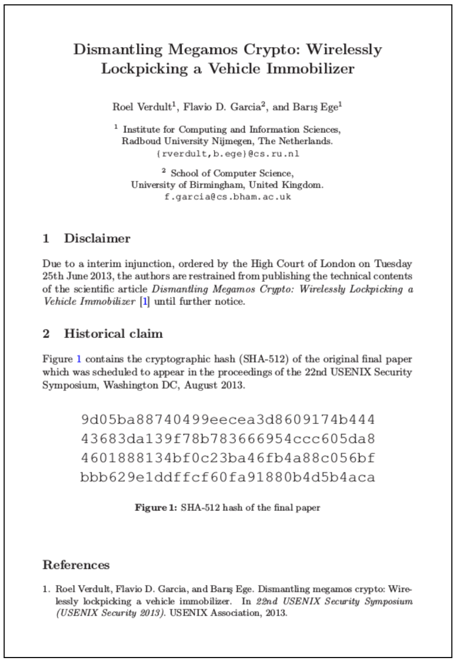
> "relating to or *characteristic* of humankind"

You could go on looking up the definition of the third non-small word in this definition and so on. But let us assume you agreed with Bob to stop after three iterations with the third non-article word in the last definition, that is *or*. Now, instead of sending to Bob the solution *folio*, you send to him *characteristic*.

How can Bob verify that you know the solution? Well, once he solved it himself, he can use the dictionary and follow the same "trail" as you did. If the final word agrees with what you had sent him, he must infer you knew the solution earlier than him. This protocol works like a one-way hash function because *characteristic* does not give any hint as to what was the first word was. I leave you to think why this protocol avoids small words?

After Bob found his solution and verified that according to the protocol it "maps" also to *characteristic*, can he be entirely sure no cheating is going on? Not with 100% certainty. It could have been possible that he was given *characteristic* as the word, but it derived from a different word. This might seem very unlikely, but at least theoretical it is a possibility. Protocols based on zero-knowledge proofs will produce a similar result—they give an answer that

[1] Let's say the, a, an, or, and …fall into the category of small words.



Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer

Roel Verdult[1], Flavio D. Garcia[2], and Barış Ege[1]

[1] Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.
{rverdult,b.ege}@cs.ru.nl

[2] School of Computer Science,
University of Birmingham, United Kingdom.
f.garcia@cs.bham.ac.uk

1 Disclaimer

Due to a interim injunction, ordered by the High Court of London on Tuesday 25th June 2013, the authors are restrained from publishing the technical contents of the scientific article *Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer* [1] until further notice.

2 Historical claim

Figure 1 contains the cryptographic hash (SHA-512) of the original final paper which was scheduled to appear in the proceedings of the 22nd USENIX Security Symposium, Washington DC, August 2013.

```
9d05ba88740499eecea3d8609174b444
43683da139f78b783666954ccc605da8
4601888134bf0c23ba46fb4a88c056bf
bbb629e1ddffcf60fa91880b4d5b4aca
```

Figure 1: SHA-512 hash of the final paper

References

1. Roel Verdult, Flavio D. Garcia, and Barış Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *22nd USENIX Security Symposium (USENIX Security 2013)*. USENIX Association, 2013.

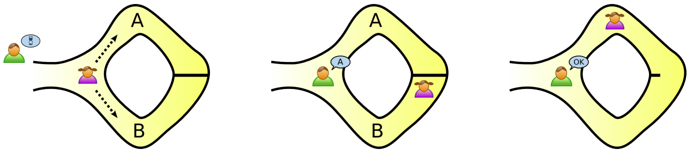
Figure 1: The authors of this paper used a hash in order to prove later that they have managed to break into cars.

might be erroneous in a very small number of cases. The point is to iterate them long enough so that the theoretical possibility of cheating is negligibly small.

By the way, the authors of the paper "Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer" who were barred from publishing their results used also a hash to prove they did the work and (presumably) managed to get into cars without a key; see Figure 1. This is very similar to the method above about crosswords: They like to prove that they did the work, but not giving out the "solution". But this also shows what the problem with such a method is: yes, we can hide the secret temporarily, but if somebody else wants to verify it, then the secret has to be made public. Bob needs to know that *folio* is the solution before he can verify the claim of Alice that she had the solution first. Similarly with the car-crypto paper: we needed to wait until September 2015 when the authors were finally able to publish their findings in order to verify the hash. Zero-knowledge proofs, in contrast, can be immediately checked, even if the secret is not public yet and perhaps never will be.

ZKP: An Illustrative Example

The idea behind zero-knowledge proofs is not very obvious and will surely take some time for you to digest. Therefore let us start with a simple illustrative example. The example will not be perfect, but hopefully explain the gist of the idea. The example is called Alibaba's cave, which graphically looks as follows:[2]



Step 1 Step 2 Step 3

Let us take a closer look at the picture in Step 1: The cave has a tunnel which forks at some point. Both forks are connected in a loop. At the deep end of the loop is a magic wand. The point of the magic wand is that Alice knows the secret word for how to open it. She wants to keep the word secret, but wants to convince Bob that she knows it.

One way of course would be to let Bob follow her, but then he would also find out the secret. So this does not work. To find a solution, let us first fix the rules: At the beginning Alice and Bob are outside the cave. Alice goes in alone and takes either tunnel labelled A in the picture, or the other tunnel labelled B. She waits at the magic wand for the instructions from Bob, who also goes into the gave and observes what happens at the fork. He has no knowledge which tunnel Alice took and calls out (in Step 2) that she should emerge from tunnel A, for example. If she knows the secret for opening the wand, this will not be a problem for Alice. If she was already in the A-segment of the tunnel, then she just comes back. If she was in the B-segment of the tunnel she will say the magic word and goes through the wand to emerge from A as requested by Bob.

Let us have a look at the case where Alice cheats, that is not knows the secret. She would still go into the cave and use, for example the B-segment of the tunnel. If now Bob says she should emerge from B, she is lucky. But if he says she should emerge from A then Alice is in trouble: Bob will find out she does not actually know the secret. So in order to fool Bob she needs to anticipate his call, and already go into the corresponding tunnel. This of course also does not work, since Bob can make any call he wants. Consequently in order to find out whether Alice cheats, Bob just needs to repeat this protocol many times. Each time Alice has a chance of $\frac{1}{2}$ to be lucky or being found out. Iterating this n times means she must be right every time and when cheating: the probability for this is $\frac{1}{2}^n$, number that for already relatively small n, say 10, is incredibly small.

There are some interesting observations we can make about Alibaba's cave and the ZKP protocol between Alice and Bob:

- **Completeness** If Alice knows the secret, Bob accepts Alice "proof" for sure. There is no error possible in that Bob thinks Alice cheats when she actually knows the secret.

[2] The example is taken from an article titled "How to Explain Zero-Knowledge Protocols to Your Children" available from http://pages.cs.wisc.edu/~mkowalcz/628.pdf.

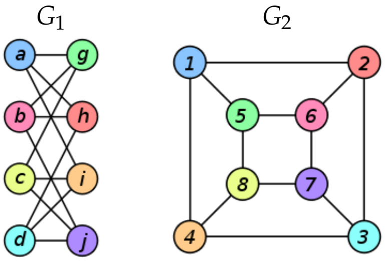
- **Soundness** If Alice does not know the secret, Bob accepts her "proof" with a very small probability. If, as in the example above, the probability of being able to hide cheating is $\frac{1}{2}$ in each round it will be $\frac{1}{2}^n$ after n-rounds, which even for small n say > 10 is very small indeed.
- **Zero-Knowledge** Even if Bob accepts the proof by Alice, he cannot convince anybody else.

The last property is the most interesting one. Assume Alice has convinced Bob that she knows the secret and Bob filmed the whole protocol with a camera. Can he use the video to convince anybody else? After a moment of thought, you will agree that this is not the case. Alice and Bob might have just made it all up and colluded by Bob telling Alice beforehand which tunnel he will call. In this way it appears as if all iterations of the protocol were successful, but they prove nothing. If another person wants to find out whether Alice knows the secret, he or she would have to conduct the protocol again. This is actually the formal definition of a zero-knowledge proof: an independent observer cannot distinguish between a real protocol (where Alice knows the secret) and a fake one (where Bob and Alice colluded).

Using an Graph-Isomorphism Problem for ZKPs

Now the question is how can we translate Alibaba's cave into a computer science solution? It turns out we need an NP problem for that. The main feature of an NP problem is that it is computational very hard to generate a solution, but it is very easy to check whether a given solution indeed solves the problem at hand.[3]

One NP problem is the *graph isomorphism problem*. It essentially determines whether the following two graphs, say G_1 and G_2, can be moved and stretched so that they look exactly the same.



Graph G_1	Graph G_2
a	1
b	6
c	8
d	3
g	5
h	2
i	4
j	7

The table on the right gives a mapping of the nodes of the first graph to the nodes of the second. With this mapping we can check: node a is connected in G_1 with g, h and i. Node a maps to node 1 in G_2, which is connected to 2, 4 and 5, which again correspond via the mapping to h, i and g respectively. Let us write σ for such a table and let us write

[3] The question whether $P = NP$ or not, we leave to others to speculate about.

$$G_1 = \sigma(G_2)$$

for two isomorphic graphs (σ being the isomorphism). It is actually very easy to construct two isomorphic graphs: Start with an arbitrary graph, re-label the nodes consistently. Alice will need to do this frequently for the protocol below. In order to be useful, we therefore would need to consider substantially larger graphs, like with thousand nodes.

Now the secret which Alice tries to hide is the knowledge of such an isomorphism σ between two such graphs. But she can convince Bob whether she knows such an isomorphism for two graphs, say G_1 and G_2, using the same idea as in the example with Alibaba's cave. For this Alice and Bob must follow the following protocol:

1. Alice generates an isomorphic graph H which she sends to Bob (in each iteration she needs to generate a different H).

2. After receiving H, Bob asks Alice either for an isomorphism between G_1 and H, or G_2 and H.

3. Alice and Bob repeat this procedure n times.

In Step 1 it is important that Alice always generates a fresh isomorphic graph. I let you think what would happen if Alice sends out twice the same graph H.

As said before, this is relatively easy to generate by consistently relabelling nodes. If she started from G_1, Alice will have generated

$$H = \sigma'(G_1) \tag{1}$$

where σ' is the isomorphism between H and G_1. But she could equally have started from G_2. In the case where G_1 and G_2 are isomorphic, if H is isomorphic with G_1, it will also be isomorphic with G_2, and vice versa.

After generating H, Alice sends it to Bob. The important point is that she needs to "commit" to this H, therefore this kind of zero-knowledge protocols are called *commitment protocols*. Only after receiving H, Bob will make up his mind about which isomorphism he asks for—whether between H and G_1 or H and G_2. For this he could flip a coin, since the choice should be as unpredictable for Alice as possible. Once Alice receives the request, she has to produce an isomorphism. If she generated H as shown in (1) and is asked for an isomorphism between H and G_1, she just sends σ'. If she had been asked for an isomorphism between H and G_2, she just has to compose her secret isomorphism σ and σ'. The main point for the protocol is that even knowing the isomorphism between H and G_1 or H and G_2, will not make the task easier to find the isomorphism between G_1 and G_2, which is the secret Alice tries to protect.

In order to make it crystal clear how this protocol proceeds, let us give a version using our more formal notation for protocols:

0)	$A \rightarrow B :$	G_1 and G_2
1a)	$A \rightarrow B :$	H_1
1b)	$B \rightarrow A :$	produce isomorphism $G_1 \leftrightarrow H_1$? (or $G_2 \leftrightarrow H_1$)
1c)	$A \rightarrow B :$	requested isomorphism
2a)	$A \rightarrow B :$	H_2
2b)	$B \rightarrow A :$	produce isomorphism $G_1 \leftrightarrow H_2$? (or $G_2 \leftrightarrow H_2$)
2c)	$A \rightarrow B :$	requested isomorphism
	$\ldots$	

As can be seen the protocol runs for some agreed number of iterations. The H_i Alice needs to produce, need to be all distinct. I hope you now know why?

It is also crucial that in each iteration, Alice first sends H_i and then Bob can decide which isomorphism he wants: either $G_1 \leftrightarrow H_i$ or $G_2 \leftrightarrow H_i$. If somehow Alice can find out before she committed to H_i, she can cheat. For this assume Alice does *not* know an isomorphism between G_1 and G_2. If she knows which isomorphism Bob will ask for she can craft H in such a way that it is isomorphism with either G_1 or G_2 (but it cannot with both). Then in each case she would send Bob a correct answer and he would come to the conclusion that all is well. I let you also answer the question whether such a protocol run between Alice and Bob would convince a third person, say Pete.

Since the interactive nature of the above PKZ protocol and the correct ordering of the messages is so important for the "correctness" of the protocol, it might look surprising that the same goal can also me achieved in a completely offline manner. By this I mean Alice can publish all data at once, and then at a later time, Bob can inspect the data and come to the conclusion whether or not Alice knows the secret (again without actually learning about the secret). For this Alice has to do the following:

1. Alice generates n isomorphic graphs $H_{1..n}$ (they need to be all distinct)

2. she feeds the $H_{1..n}$ into a hashing function (for example encoded as as matrix)

3. she takes the first n bits of the output of the hashing function: whenever the output is 0, she shows an isomorphism with G_1; for 1 she shows an isomorphism with G_2

The reason why this works and achieves the same goal as the interactive variant is that Alice has no control over the hashing function. It would be computationally just too hard to assemble a set of $H_{1..n}$ such that she can force where 0s and 1s in the hash values are such that it would pass an external test. The point is that Alice can publish all this data on the comfort of her own web-page, for example, and in this way convince everybody who bothers to check.

The virtue of the use of graphs and isomorphism for a zero-knowledge protocol is that the idea why it works are relatively straightforward. The disadvantage is that encoding any secret into a graph-isomorphism, while possible, is awkward. The good news is that in fact any NP problem can be used as part of a ZKP protocol.

Using Modular Logarithms for ZKP Protocols

While information can be encoded into graph isomorphisms, it is not the most convenient carrier of information. Clearly it is much easier to encode information into numbers. Let us look at zero-knowledge proofs that use numbers as secrets. For this the underlying NP-problem is to calculate discrete logarithms. It can be used by choosing public numbers A, B, p, and private x such that

$$A^x \equiv B \ mod \ p$$

holds. The secret Alice tries to keep secret is x. The point of the modular logarithm is that it is very hard from the public data to calculate x (for large primes). Now the protocol proceeds in three stages:

- **Commitment Stage**

  1. Alice generates z random numbers $r_1, \ldots, r_z$, all less than $p-1$. Alice then sends Bob for all $i = 1, \ldots, z$:

     $$h_i = A^{r_i} \ mod \ p$$

  2. Bob generates z random bits, say $b_1, \ldots, b_z$. He can do this by flipping z times a coin, for example.
  3. For each bit b_i, Alice sends Bob an s_i where

     $$\begin{array}{ll} \text{if } b_i = 0\text{:} & s_i = r_i \\ \text{if } b_i = 1\text{:} & s_i = (r_i - r_j) \ mod \ (p-1) \end{array}$$

     where r_j is the lowest j where $b_j = 1$.

For understanding the last step, let z be just 4. We have four random values r_i chosen by Alice and four random bits b_i chosen subsequently by Bob, for example

$$\begin{array}{rcccc} r_i\text{:} & 4 & 9 & 1 & 3 \\ b_i\text{:} & 0 & 1 & 0 & 1 \\ & & \uparrow & & \\ & & j & & \end{array}$$

The highlighted column is the lowest where $b_i = 1$ (counted from the left). That means $r_j = 9$. The reason for letting Alice choose the random numbers $r_1, \ldots, r_z$ will become clear shortly. Next is the confirmation phase where Bob essentially checks whether Alice has sent him "correct" s_i and h_i.

- **Confirmation Stage**

  1. For each b_i Bob checks whether s_i conform to the protocol

     $$\begin{array}{ll} \text{if } b_i = 0\text{:} & A^{s_i} \overset{?}{\equiv} h_i \ mod \ p \\ \text{if } b_i = 1\text{:} & A^{s_i} \overset{?}{\equiv} h_i * h_j^{-1} \ mod \ p \end{array}$$

To understand the case for $b_i = 1$, you have to do the following calculation:

$$\begin{aligned} A^{s_i} &= A^{r_i - r_j} \\ &= A^{r_i} * A^{-r_j} \\ &= h_{r_i} * h_{r_j}^{-1} \ mod \ p \end{aligned}$$

What is interesting that so far nothing has been sent about x, which is the secret Alice has. Also notice that Bob does not know r_j. He received

$$r_j - r_j, r_m - r_j, \ldots, r_p - r_j \ mod \ p - 1$$

whenever his corresponding bits were 1. So Bob does not know r_j and also does not know any r_i where the bit was 1. Information about the x is sent in the next stage (obviously not revealing x).

- **Proving Stage**

  1. Alice proves she knows x, the discrete log of B, by sending

  $$s_{z+1} = x - r_j \ mod \ p - 1$$

  2. Bob confirms

  $$A^{s_{z+1}} \overset{?}{\equiv} B * h_j^{-1} \ mod \ p$$

To understand the last step, you have to do the trick again that

$$A^{s_{z+1}} = A^{x - r_j} = \ldots$$

which I leave to you.

Now the question is how can Alice cheat? In order to cheat she has to coordinate what she sends as h_i in step 1 and s_i in step 3 of the commitment stage, and also what to send as s_{z+1} in the proving stage. For the latter of course Alice does not know x, so she just chooses some random number for s_{z+1} and calculates

$$A^{s_{z+1}}$$

and then solves the equation

$$A^{s_{z+1}} \equiv B * y \ mod \ p$$

for y. This is easy since no logarithm needs to be computed. If Alice can guess the j where the first 1 will appear in Bob's bit vector, then she sends the inverse of y as h_j and 0 as s_j. However, notice that when she calculates a solution for y she does not know r_j. For this she would need to calculate the modular logarithm

$$y \equiv A^{r_j} \ mod \ p$$

which is hard (see step 1 in the commitment stage).

Having settled on what h_j should be, now what should Alice send as the other h_i and other s_i? If the b_i happens to be a 1, then the h_i and other s_i need to satisfy the test

$$A^{s_i} \overset{?}{\equiv} h_i * h_j^{-1} \ mod \ p$$

where she has already settled on the value of h_j^{-1}. Lets say she choses s_i at random, then she just needs to solve

$$A^{s_i} \equiv z * h_j^{-1} \ mod \ p$$

for z. Again that is easy, but it does not allow us to know r_i, because then we would again need to solve a modular logarithm problem. Let us call an h_i which was solved the easy way as *bogus*. Alice has to produce bogus h_i for all bits that are going to be 1 in advance! This means she has to guess all the bits correctly. (Yes? I let you think about this.)

Let us see what happens if she guesses wrongly: Suppose the bit $b_i = 1$ where she thought she will get a 0. Then she has already sent an h_i and h_j and now must find an s_i such that

$$A^{s_i} \equiv h_i * h_j^{-1} \ mod \ p$$

holds. For this remember in calculating h_i, she just chose a random s_i. Now she has to send a genuine one. But this is of course too hard. If she knew the genuine r_i and r_j for h_i and h_j, it would be easy (in this case $s_i = r_i - r_j$). But she does not. So she will be found out. If $b_i = 0$, but she thought she will get a 1, then she has to send a s_i which satisfies

$$A^{s_i} \equiv h_i \ mod \ p$$

Again she does not know r_i. So it is a too hard task and she will be found out again.

To sum up, in order for Alice to successfully cheat Bob, she needs to guess *all* bits correctly. She has only a $\frac{1}{2^z}$ chance of doing this.

Further Reading

Make sure you understand what NP problems are.[4] They are the building blocks for zero-knowledge proofs. Zero-Knowldege proofs are not yet widely used in production systems, but it is slowly gaining ground. One area of application where they pop up is crypto currencies (for example Zerocoins or how to make sure a Bitcoin exchange is solvent without revealing its assets).

If you want to brush up on the modular logarithm problem, the Khan Academy has a nice video:

[4] http://en.wikipedia.org/wiki/NP_(complexity)

https://www.khanacademy.org/video/discrete-logarithm-problem