26
The Laws of Cryptography
Zero-Knowledge Protocols

26.1 The Classes NP and NP-complete.

26.2 Zero-Knowledge Proofs.

26.3 Hamiltonian Cycles.

An *NP*-complete problem known as the *Hamiltonian Cycle Problem* gives a good illustration of a simple zero-knowledge proof. The problem starts with an *undirected graph* $G = (V, E)$, where V is a finite set of *vertices*, E is a set of ordered pairs of vertices forming *edges*, and if (u, v) is an edge, so is (v, u). A *path* in G is a sequence of vertices $v_0, v_1, v_2, \ldots, v_{n-1}$ such for each $0 \leq i \leq n-2$, (v_i, v_{i+1}) is an edge. A path is a *cycle* (ends where it starts) if $v_{n-1} = v_0$. A path is *simple* (doesn't cross itself) if no vertex appears more than once. A path is *complete* if it includes every vertex. The *Hamiltonian Cycle Problem* (**HC**) asks if a given graph has a simple complete cycle. It turns out that **HC** is an *NP*-complete problem, so in general no efficient algorithm is known.

If one had an efficient algorithm to solve **HC**, then one would also have an efficient algorithm to actually obtain the Hamitonian cycle itself. First check the entire graph to see if there is such a cycle. Then try deleting each edge in turn, checking again if there is still a Hamilintonian cycle, until only the edges of a Hamiltonian cycle remain. (There may be more than one such cycle, but this method will find one of them.)

For a given graph, even a large one, it may be easy to decide this problem, but there is no known efficient algorithm to solve the general problem as the number of vertices increases.

Consider now the specific simple graph in Figure 26.1. The graph illustrated in this figure is the same as the vertices and edges of a dodecahedron (a 12-sided regular polyhedron with each side a pentagon). All that is present is a wire framework of the edges, and the framework has been opened up from one side and squashed onto the plane.

This graph is not complicated, but it still takes most people at least a minute or two to find one of the Hamiltonian cycles in the graph. Try to do it now, without peeking ahead (and without writing in the book). Once you have found a cycle, read on. The cycle is shown later in Figure 26.2 as a thicker set of lines.

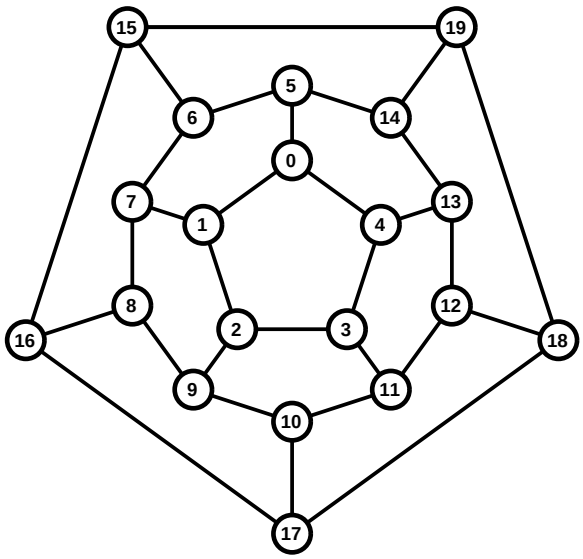

Figure 26.1 Vertices and Edges of a Dodecahedron.

Dodecahedron Graph		
Vertex	**Edges**	**Cycle**
0	1, 4, 5	1
1	0, 2, 7	7
2	1, 3, 9	3
3	2, 4, 11	4
4	0, 3, 13	0
5	0, 6, 14	14
6	5, 7, 15	5
7	1, 6, 8	8
8	7, 9, 16	16
9	2, 8, 10	2
10	9, 11, 17	9
11	3, 10, 12	10
12	11, 13, 18	11
13	4, 12, 14	12
14	5, 13, 19	13
15	6, 16, 19	6
16	8, 15, 17	17
17	10, 16, 18	18
18	12, 17, 19	19
19	14, 15, 18	15

Table 26.1 Dodecahedron Graph.

In computer algorithms, graphs are often represented as a list of vertices, and for each vertex, a list of the vertices which together with the first one make up an edge. Suppose that the vertices are numbered from 0 to $n-1$, or in this case, to 19. Table 26.1 gives information describing the graph in this way. The rightmost column also shows a Hamiltonian cycle by giving, for each vertex, the next vertex in the cycle.

Now on to a zero-knowledge proof by Alice to Bob that she has a Hamiltonian cycle of this graph while revealing nothing about the cycle itself.

Alice must carry out a number of steps in the process of a probablistic proof. After n stages, Bob can be almost certain that Alice has a Hamiltonian cycle for the graph, with only a probability of 2^{-n} that she does not have a cycle, but is cheating. At each stage Alice chooses a new (true) random permutation of the vertices in the graph. In the example below, we assume she has chosen the rearrangement of vertices given in Table 26.2 .

Alice rearranges the table by sorting the newly renumbered vertices into order again. For each vertex, the list of vertices at the other end of an edge must be made all the same length, using extra dummy entries. (In the case here all the lists are already the same length.) Finally

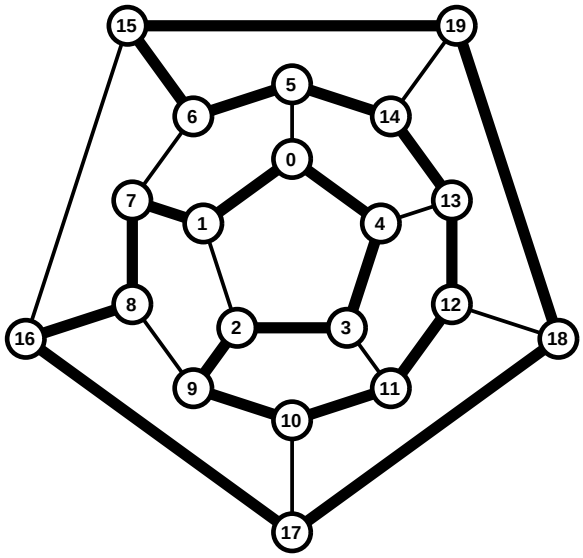

Figure 26.2 Hamiltonian Cycle on a Dodecahedron.

Original vertex:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Changed to:	12	17	16	19	8	13	7	3	10	5	18	11	2	9	15	6	0	14	4	1

Table 26.2 Permutation of Vertices.

Permuted Dodecahedron Graph			
Vertex	**Edges**	**Cycle**	**Permutation**
0	**14**, 6, 10	**14**	16
1	**6**, 4, 15	**6**	19
2	**11**, 4, 9	**11**	12
3	**10**, 7, 17	**10**	7
4	14, **1**, 2	**1**	18
5	**16**, 10, 18	**16**	9
6	0, 1, **7**	**7**	15
7	3, **13**, 6	**13**	6
8	**12**, 9, 19	**12**	4
9	15, **2**, 8	**2**	13
10	5, 3, **0**	**0**	8
11	19, 2, **18**	**18**	11
12	**17**, 8, 13	**17**	0
13	12, 7, **15**	**15**	5
14	18, **4**, 0	**4**	17
15	1, 13, **9**	**9**	14
16	**19**, 17, 5	**19**	2
17	12, 16, **3**	**3**	1
18	11, **5**, 14	**5**	10
19	16, **8**, 11	**8**	3

Table 26.3 Permuted Dodecahedron Graph.

she must randomize the order of these lists. Table 26.3 shows the results of all these changes.

Alice needs a means to selectively reveal parts of this last table, but she has to commit to the entire table without being able to cheat. This is easy to do with any cryptosystem, such as the AES, for example. For each item in the table needing concealment, use a separate encryption with a separate key. (Or similar items can be handled with a single key, as long as an extra random string is included to foil ciphertext comparisons.) Alice sends all the information in the table, with each item encrypted. Notice that if Alice encrypts the vertex number 17 with some AES key, it is not feasible for her to find another key that decrypts the same ciphertext to some other number, say 16, assuming there are a large number of redundant bits in the ciphertext.

So Alice sends the entire table in permuted and encrypted form to Bob. Bob then asks Alice to show him one of two parts of the table:

- the original graph, or
- the Hamiltonian cycle.

In the second case, Alice sends the keys needed to reveal only the parts of the table in **boldface**.

(The vertex numbers are implicit, starting at 0.) This shows each succesive vertex in the cycle, and shows that the vertex is actually on an edge. All this is assuming that the hidden graph is really the correct one. If Alice wanted to cheat, it would be easy for her to make up numbers so that it would look like a Hamiltonian cycle was present.

In the second case, Alice sends the keys needed to reveal all parts of the table except for the column labeled "cycle". The extra column labeled "permutation" is there to make it easy for Bob to verify that the permuted graph is still the same as the original. (There are no efficient algorithms to show that two graphs are really the same.)

If Alice doesn't know a cycle, she can cheat and answer either question correctly, but she has to commit herself to an answer before knowing what part Bob will ask for. In the first case she reveals nothing at all about how the Hamiltonian cycle is situated in the graph, but only that it exists. In the second case she reveals nothing at all about the Hamiltonian cycle. At each stage of this probablistic proof, Alice chooses a new permutation of the vertices, rearranges the table again, and chooses new keys for encrypting. At each step, if she is cheating, she will caught half the time on the average.

Alice could conceivably use **HC** to establish her identity. First she constructs a large, complex graph G, making sure at each stage of the construction that G has a Hamiltonian cycle. Alice securely transfers the graph G to a trusted server T. Then if Alice wanted to prove her identity to Bob, she could have Bob retrieve the graph from T, and then she could prove (probablistically, in zero knowledge) that she has a Hamiltonian cycle. Even an eavesdropper would not learn the Hamiltonian cycle. This particular method has two separate flaws as a way for Alice to identify herself:

- The method is subject to replay attacks, where Boris just sends someone else exactly the same messages Alice transmits as a way to "prove" that he is Alice.
- It is necessary for Alice to create a "hard" instance of **HC**, that is, an instance where known algorithms take a very long time to find a Hamiltonian cycle.

Alice could handle the first flaw by adding a timestamp to the part of the graph that she encrypts. The second flaw above is the difficult one, since finding hard instances of an *NP*-complete problem is often a difficult computation itself, and that especially seems to be the case with **HC**. The next chapter shows how to use factoring as the hard problem, and factoring has been studied much more extensively than **HC**.

26.4 Other Zero-Knowledge Proofs.