Scribe Notes

1 Overview

Today's class will go over three topics:

- Reminders: Hamilton Cycle and Zero-Knowledge Protocols
- Visual Cryptography (closely related to one-time pad)
- Zero-knowledge subset sum This is a variant of subset sum, that is used in cryptography, requiring n integers of size n bits.

  The normal subset sum problem takes as input a tuple of numbers $a = (a_1, a_2, ..., a_n)$ and an integer bound A. It either outputs a bit vector $x = x_1x_2 \cdots x_n$ such that $x_i \in \{0, 1\}$, and $\sum_{i=1}^{n} x_i a_i = A$ if one exists, or returns "none".

  The variant we will discuss further requires that n is even, $\sum_{i=1}^{n} x_i = n/2$ and that each of the a_i is represented as an n-bit integer.

- Sources handed out in class:

  Visual Cryptography: http://www.cacr.math.uwaterloo.ca/~dstinson/visual.html

  (Also, http://citeseer.ist.psu.edu/naor95visual.html is a good paper on Visual Cryptography, though it was not distributed in class).

  One time pad: http://en.wikipedia.org/wiki/One-time_pad

2 Review and thoughts about last lecture

2.1 Hamilton Path/Cycle

Last time, we went over a zero-knowledge proof of Hamilton cycle. For yourself, construct a zero-knowledge proof of Hamilton path. There is some overlap between the proofs, and some key differences.

2.2 Algorithms versus Protocols

An algorithm is a set of instructions that are followed faithfully (presumably by a computer)

A protocol is a set of instructions that may or may not be followed faithfully. Protocols are usually designed for multiple agents, and the agents can attempt to cheat. Promises are made only to those who follow the protocol. If the protocol is not followed, no promises are made–an agent may win, or lose.

2.3 Grey Paint Clarification

Informally, let us think of "grey paint" as the equivalent of the grey paint on lottery scratch tickets, where what is underneath the paint is obscured until the paint is scratched off. In the proof of Hamilton Cycle, we used a concept of "grey paint" as something that could obscure private data from the Verifier, but could be "scratched off" selectively to reveal only pieces at a time. Grey paint will be replaced by a "bit commitment" scheme later on in the course. Search for "bit commitment" to learn more.

3 Visual Cryptography

3.1 Terminology

The purpose of visual cryptography is to provide a simple way to securely encrypt graphical images. The end result does not require intensive mathematical computations; instead, it can be done visually, using two transparencies that are overlaid. Since the goal is to encrypt a graphical image that can be visually decrypted, it makes sense to use images in the encryption themselves (see

http://citeseer.ist.psu.edu/naor95visual.html for more information). For the purpose of this discussion, lets assume that we are encrypting black and white images, so that each pixel can be represented as being either black or white.

We are going to talk about "ying/yang" graphical bits, which we will call yits. Yits will be used as our building blocks for visual encryption, in a similar way that we normally use bits in binary computations. Any two images where each image occupies half the area and their overlay fills the area can be called yits. (See table below for examples, using a circle instead of a square area).

ying	yang	filled cell	open cell
★	✪	●	○

Key to visual cryptography symbols

Let us call ★the "ying" yit and ✪the "yang" yit. One important property of these yits is how they behave when they are overlapped. Drawing a ying on top of a ying looks like a ying. Similarly, drawing a yang on top of a yang is simply a yang. However, if a ying is laid down on a yang, the result is a filled circle. See table below for graphical illustration.

ying + ying	★+ ★= ★
yang + yang	✪+ ✪= ✪
ying + yang	★+ ✪= ●
yang + ying	✪+ ★= ●

Addition of yits

3.2 Problem Statement

We will now consider the problem of encrypting a message M. M could be thought of as a rectangular picture with $m \times n$ pixels, where each pixel is either solid black or solid white. Equivalently, M can be thought of as a matrix of size $m \times n$ where each cell is either 0 or 1.

Alice has access to M and wants to graphically encrypt it before sending to Bob. Suppose, in addition, Alice has access to a random matrix $R_{m \times n}$, which was constructed as follows.

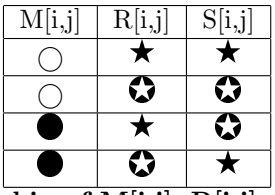
1. For each cell in the array R, Alice tosses a fair coin, equally likely heads or tails.
2. If it lands heads, the cell is filled in with a ying. If it lands tails, it is filled in with a yang.
3. The fair coin is tossed mn times, once for each cell in R, and each coin flip is independent. R has absolutely no information about M, and is created independently from M.

After constructing R, Alice transmits R to Bob using a secure private channel, so that both Alice and Bob have access to R. Now, Alice can follow a protocol to generate a new visual message S, with the property that S combined with R allows you to see M. Furthermore, S can be transmitted along a public channel, without an adversary being able to discern M.

This concept is similar to a one-time pad. Think of R as a page on the one-time pad, and M as the message being encrypted. Then S represents the encrypted plaintext, and R the decryption key. The protocol presented is both simple and secure as long as a private channel for communication is available. However, the protocol presented has the property that for every message yit in M, Alice will have to transmit two to Bob (one in R, and another in S). The protocol follows.

1. Alice constructs an array $S_{m \times n}$:
   (a) For each cell in M,
   (b) If M[i,j] is empty, use R[i,j]'s yit (S[i,j] = R[i,j]).
   (c) If M[i,j] is filled in (●), flip R[i,j]'s yit

This is shown graphically below:



Relationship of M[i,j], R[i,j] and S[i,j]

2. Alice sends S to Bob.

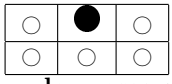
3. To reconstruct the message M, Bob simply needs to overlay S and R. Whenever a 0 or open space is encoded, R[i,j] + S[i,j] = R[i,j] = S[i,j]. On the other hand, whenever a 1 or filled space is encoded, R[i,j] + S[i,j] will be a filled space.

   A graphical table showing this is below, based loosely on the one located in

   `http://www.cacr.math.uwaterloo.ca/~dstinson/visual.html`

Results of R[i,j] + S[i,j], for all possibly combinations of M[i,j], R[i,j] and S[i,j]

A small example:
M is constructed as an array with a 1 in the second column in the first row, and 0s everywhere else. In its graphical version, that means the only filled cell is located in the second column of the first row.



Example message M

R is constructed at random, as the result of a fair coin toss, and transmitted over a secure private channel to Bob, so that both Alice and Bob agree on R. R is constructed in advance.

Example random R

Now, we construct S, by looking at M. M is only filled in at the second column of the first row, so we flip that yit and leave the rest the same as in R. Alice can now send S to Bob over a public channel, without anyone being able to decipher M from S.

Example of S construction, based on R and M

When Bob overlays S and R, he gets

Results of R + S

Bob can easily read M out of this combination: any time Bob sees a filled in circle, he knows there was originally a filled in circle; any time the circle is not filled in, then it was originally empty. Thus, M can be reconstructed.

The concept of visual cryptography is strongly related to one-time pads, especially if we consider generating multiple random matrices like R in advance, and transmitting them over a private channel. Each can only be used once, and is discarded after use, just like the pages on a one-time pad. Sources have been referenced for both visual cryptography and one-time pads.

3.3 Is S random?

An interesting question that arises from this problem is whether S is random. We define random as occurring with probability $\frac{1}{2^{nm}}$. It is clear that this is true for R, as it is constructed by flipping a coin mn times, and each coin toss has a probability 1/2 of being either heads or tails. Let us consider two proofs. One will show that S is not random, and the other that S is random. Clearly, only one can be correct.

3.3.1 S is not random

Proof. R is random. R with S gives M. Therefore, M is in S (and S is not random). □

3.3.2 S is random

Proof. There is a 1-1 correspondence between all 2^{mn} possible Rs and 2^{mn} possible Ss. Each of the 2^{mn} possibilities for R is equally likely. The same thing is true for S. Thus, all 2^{mn} possible Ss are equally likely, and S is random. □

3.3.3 Evaluating the proofs

The first proof has a fallacy. S is random and does not contain M. However, S combined with R does contain M, since S depends on R. The two assumptions of the first proof are true. R is random, and R with S gives M. While S being non-random (say, S=M) is one way to achieve R with S giving M, it is not the only way. In fact, since S depends on R and M, S and R together can provide more information than either one of S or R alone.

An illuminating example of the logical fallacy: When people have two eyes, they have binocular vision. When you remove the left eye, you lose binocular vision. Therefore, binocular vision is located in the left eye. This argument is clearly wrong, and it fails for the same reason the proof that S is not random fails. While the loss of the left eye contributes to the loss of binocular vision, it does not mean that binocular vision is located in the left eye.

4 Zero Knowledge Example: Subset Sum Variant

We are now going to return to zero knowledge proofs. Last lecture, we reviewed Hamilton cycle. Another example of a zero knowledge proof is presented here as a variant of the subset sum problem.

4.1 Review of Subset Sum

The normal subset sum problem can be formalized as an algorithm:

INPUT: An ordered list of numbers $a = (a_1, a_2, ..., a_n)$, and a positive integer bound A.

OUTPUT: $\begin{cases} \text{A bit vector } x = x_1 x_2 \cdots x_n \text{ such that } x_i \in \{0,1\}, \text{ and } \sum_{i=1}^{n} x_i a_i = A \text{ if x exists} \\ \text{“NONE” otherwise} \end{cases}$

The subset sum problem as defined is NP-complete, and is also an important problem in cryptography.

4.2 Our variant

We will present a zero-knowledge proof for a variant of subset sum. This variant differs from classical subset sum in a few ways:

- n is a positive, even integer.
- Each a_i is a n-bit integer.
- $\sum_{i=1}^{n} x_i = n/2$
  In other words, exactly half of the values will contribute to the sum, and half will not.
- Addition is done mod 2^n.

We will sketch an “algorithm” of sorts, that takes as input the particular instance of subset sum and outputs a protocol for the Prover and Verifier to follow. The main idea is to split the problem into pieces that the Verifier can ask for, where if all the pieces are correct it provides a proof that the Prover is telling the truth. By revealing only one piece to the Verifier, the Prover gives up evidence but not a proof of the theorem. If properly constructed, that piece gives no information that the Verifier could not have discovered on her own.

INPUT:

- n is a positive, even integer
- A fixed, ordered array of n n-bit integers of the form $a_i = a_{i1}a_{i2}\ldots a_{in}$, $1 \leq i \leq n$, where each $a_{ij} \in \{0,1\}$
- An n-bit positive integer bound $A = A_1 A_2 \ldots A_n$
- A security parameter k.

OUTPUT:
A zero-knowledge protocol for a Prover and Verifier to follow that guarantees:

- The Prover can only follow the protocol if $\exists x_1, .., x_n \in \{0,1\}$ such that $(\sum_{i=1}^{n} x_i a_i) \mod 2^n = A$ and $\sum_{i=1}^{n} x_i = n/2$.
- If the Prover follows the protocol then the Prover is guaranteed that no information will be given away to the Verifier that the Verifier cannot (efficiently) get for herself. This is true even if the Verifier is cheating.
- If the Verifier follows the protocol, then the probability that a cheating Prover (one who does not truly know $x_1, x_2, \cdots, x_n$ can fool the Verifier is at most $(\frac{3}{4})^k$.

We now describe one round of the protocol. For k rounds, simply repeat the following, k times.
PROVER
Privately:

1. Split $a_1 = r_1 + s_1$, $a_2 = r_2 + s_2$ and in general $a_i = r_i + s_i$, where each r_i is generated uniformly at random from the 2^n possibilities (from the set $\{0, ..., 2^n - 1\}$), and each s_i is chosen so that $r_i + s_i \mod 2^n = a_i$. (Remember, each of r_i and s_i are n-bit numbers that can be expressed $r_i = r_{i1}r_{i2}\ldots r_{in}$.

2. Create a permutation σ, chosen uniformly at random from the $n!$ possible permutations, and reorder the sums $r_i + s_i$ according to σ. (See Figure 4.2 for details of what is occurring).

   [Aside: Suppose $x_i = 1$. Then $r_i + s_i$ is part of the sum for A, and so is $r_{\sigma(i)} + s_{\sigma(i)}$.]

3. Let $R = \sum_{i=1}^{n} x_{\sigma(i)} r_{\sigma(i)}$.

4. Let $S = \sum_{i=1}^{n} x_{\sigma(i)} s_{\sigma(i)}$.

5. By construction, $R + S = A \mod 2^n$. The finished work that the Prover does is in Figure 4.2, and a concrete example is in Figure 4.2.

6. Define $Private_1$ as the list of $a_i = r_i + s_i$, the mapping σ, and the resulting rows of $r_{\sigma(i)} + s_{\sigma(i)}$. In other words, this is the decomposition of a into r and s, the permutation mapping σ, and the same r and s in a new order (so we can check that it is a valid decomposition, and a valid permutation). These elements are bordered in blue dashed lines in Figure 4.2.

7. Define $Private_2$ as the set of $r_{\sigma(i)}$ along with $x_{\sigma(i)}$ and R. This shows you all the random bit values that are generated, a bit vector showing which half of them are used to obtain R (since they are not in order, this cannot be used to solve subset sum) and gives R as a checksum, so you can check the addition. These elements are bordered in blue dashed lines in Figure 4.2.

8. Define $Private_3$ as the set of $s_{\sigma(i)}$ along with $x_{\sigma(i)}$ and S. This shows you another set of random values that are generated, a bit vector showing which half of them are used to obtain S and gives S as a checksum, so you can check the addition. From the earlier proof in Visual Cryptography, we know that every s_i is random, as well as S. These elements are bordered in blue dashed lines in Figure 4.2.

9. Define $Private_4$ as R and S, which the verifier can add to check A. These elements are bordered in blue dashed lines in Figure 4.2.

VERIFIER
Publicly:

1. Chooses at random, with probability 1/4, one of the private pieces that she wishes to verify.

2. Checks that the provided piece of information was constructed correctly. If it was not, reject.

3. After k iterations, if no piece of information has been rejected, accept.

If the Prover is lying, then at least one of the 4 private pieces must be inconsistent. Since only one piece of four was verified, at each point there is a 3/4 chance the prover is lying. However, as k becomes large and the verifier has accepted k independent pieces, it is less and less likely that the prover is lying and simply being undiscovered.

4.3 Analysis of Protocol

Let us recall the the three guarantees we claimed this protocol would offer.

1. The Prover can only follow the protocol if $\exists x_1, .., x_n \in \{0, 1\}$ such that $(\sum_{i=1}^{n} x_i a_i) \mod 2^n = A$ and $\sum_{i=1}^{n} x_i = n/2$.

2. If the Prover follows the protocol then the Prover is guaranteed that no information will be given away to the Verifier that the Verifier cannot (efficiently) get for herself. This is true even if the Verifier is cheating.

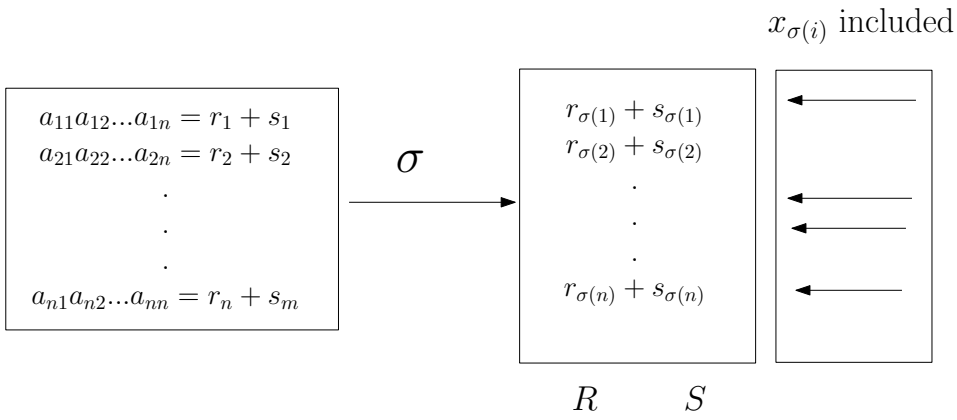


Figure 1: Result of Prover's work

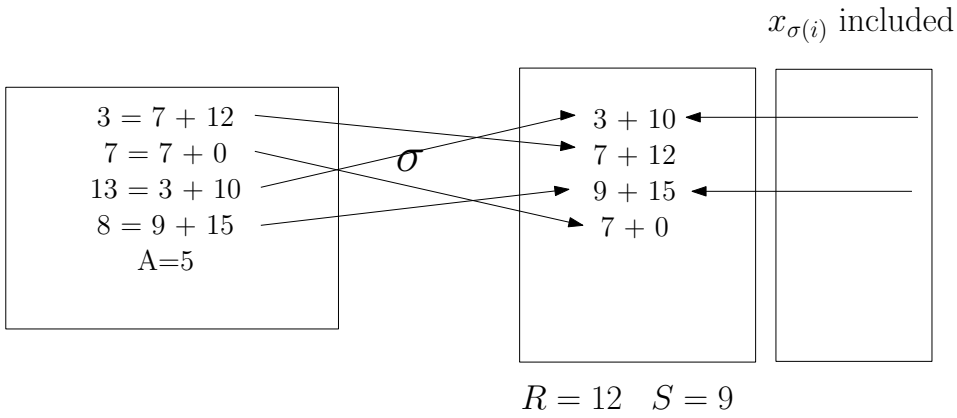


Figure 2: A concrete example of the Prover's work

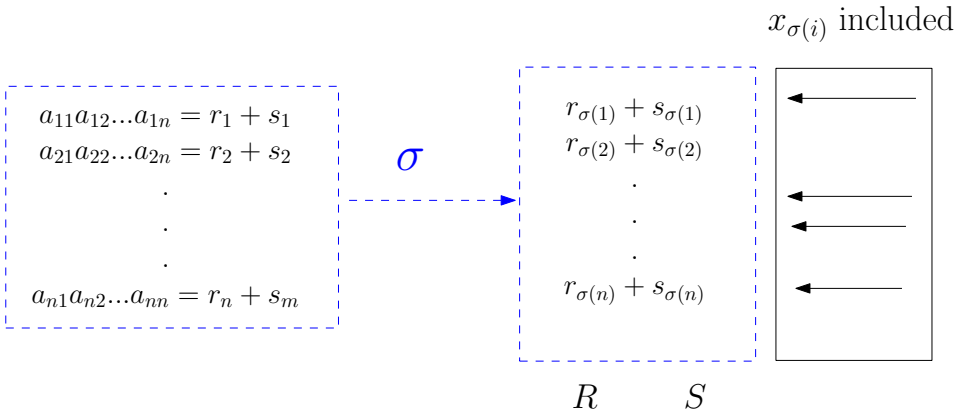


Figure 3: Private 1. Items bordered in blue dashed lines are revealed to the Verifier

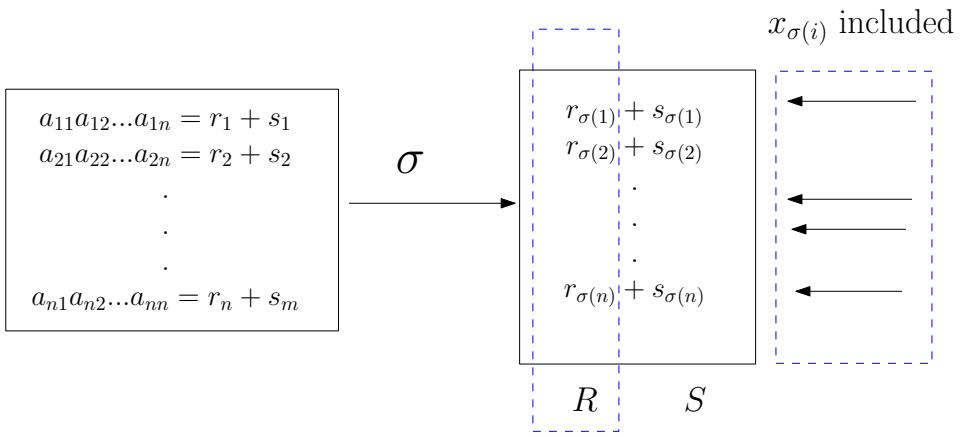


Figure 4: Private 2. Items bordered in blue dashed lines are revealed to the Verifier

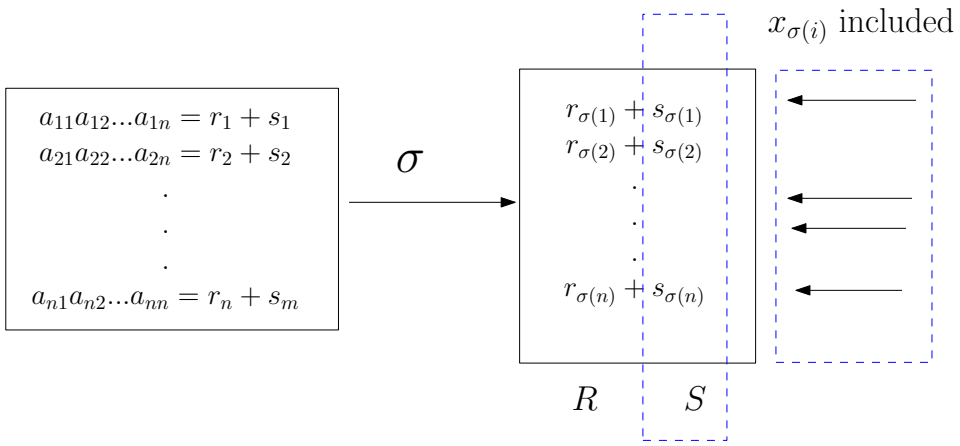


Figure 5: Private 3. Items bordered in blue dashed lines are revealed to the Verifier

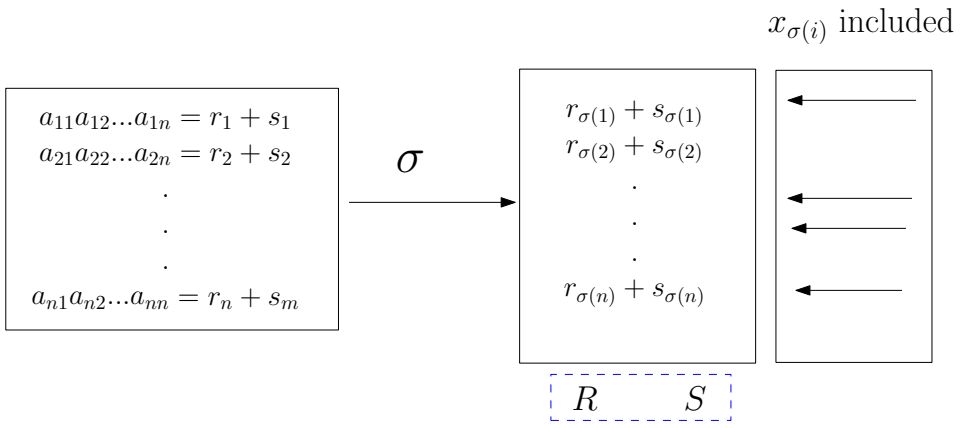


Figure 6: Private 4. Items bordered in blue dashed lines are revealed to the Verifier

3. If the Verifier follows the protocol, then the probability that a cheating Prover (one who does not truly know $x_1, x_2, \cdots, x_n$) can fool the Verifier is at most $(\frac{3}{4})^k$.

We will now analyze each claim, one at a time.

1. The Prover can only follow the protocol if $\exists x_1, .., x_n \in \{0, 1\}$ such that $(\sum_{i=1}^{n} x_i a_i) \mod 2^n = A$ and $\sum_{i=1}^{n} x_i = n/2$.

   If the prover knows an answer to the particular subset sum instance, then the prover can always answer the Verifier's questions correctly, and will always pass. However, if the prover does not know an answer to the particular instance, then at least one of the four pieces of private information must be wrong(since subset-sum is NP-complete, the Prover cannot efficiently solve the instance on the spot), and therefore, the Prover is not following the protocol exactly.

2. If the Prover follows the protocol then the Prover is guaranteed that no information will be given away to the Verifier that the Verifier cannot (efficiently) get for herself. This is true even if the Verifier is cheating.

   The set of all four pieces of information constitutes a complete proof that will pass information to the Verifier. However, each individual piece will not do so. Think of the Verifier not receiving any information that she cannot get for herself as though the Verifier could simulate, using only the publicly available numbers and a fair coin, what the Prover shows her.

   First, consider what happens if the Verifier asks for Private 1. Then all she sees is a decomposition of the (public) numbers $a = (a_1, a_2, ..., a_n)$ into sums of random numbers that have then been reordered. Since the Verifier has access to a herself, she could independently generate random numbers r_i, and numbers s_i such that $a_i = r_i + s_i$. Furthermore, she can reorder these sums of random numbers herself. So showing the Verifier only Private 1 does not impart any private information.

   Now, consider what happens if the Verifier asks for Private 2. All she sees is a list of n random numbers r_i, a sum R and a pointer to which r_i were included in the R. The Verifier can also independently generate n random numbers, choose at random n/2 of them to include in the sum, and then sum them. So no additional information is given to the Verifier by showing only Private 2.

   Showing only Private 3 to the Verifier is a symmetric proof to showing only Private 2. Again, the verifier sees a list of n random numbers s_i, a sum S, and a pointer to the n/2 s_i that were summed to get S. Once more, the Verifier can simulate this herself

   Showing only Private 4 to the Verifier involves showing that two random numbers $R + S = A \mod 2^n$. Obviously, the Verifier can also simulate the generation of a random number R, and then choose the the number S to make $R + S = A \mod 2^n$. So again, no information is given to the Verifier that she cannot get herself.

   It doesn't matter if the Verifier cheats, because during each round, she still only has access to one private piece of information, and no one piece of the information gives away anything she could not have generated herself.

3. If the Verifier follows the protocol, then the probability that a cheating Prover (one who does not truly know $x_1, x_2, \cdots, x_n$ can fool the Verifier is at most $(\frac{3}{4})^k$.

   How can a prover cheat? A cheating prover is one who does not know the bit vector x but guesses which one of the elements the Verifier will request. The cheating prover can always make sure that one of those elements is accurate. For example, it is easy to make Private 2 accurate on its own. However, all the other 3 private pieces of information cannot easily be made consistent (making all four pieces consistent implies solving the instance, which is hard since subset sum is NP-complete).

   At each stage, he has to present the Verifier at least one of the four pieces of private information. If at any time one of those pieces is rejected (it was not generated according to the protocol) then the Verifier will not pass him.

   On the other hand, suppose that the one piece of information passes for any one cycle. That means that the Prover could still have 3/4 of the private pieces constructed incorrectly. So there is probability 3/4 that the Prover is cheating after any one round.

If the Verifier follows the protocol and randomly selects which of the private pieces of information to verify, then even a cheating Prover cannot easily predict which one of the private elements to make correctly. So, after k iterations, the likelihood that the prover is cheating is at most $(\frac{3}{4})^k$.