ABSTRACT

SINKAR, GAURANG SANJEEV. A Cooperative Approach to Mitigating Sybil Attack in Wireless Sensor Networks. (Under the direction of Dr. Rudra Dutta.)

Sybil attack is one of the major threats to security of wireless sensor networks. In a Sybil attack, a compromised malicious node assumes identity of multiple nodes. By doing so, a malicious node can subvert a reputation system or gain more access to network resources than it is supposed to, or can have negative effects on redundancy mechanisms (like multipath routing) used by the network. We propose the use of monolithic computational puzzles of differential complexity to mitigate Sybil attack in wireless sensor networks. We present a solution based on cooperative action by network nodes that can be used to mitigate Sybil attack in a homogeneous (where nodes have similar computation capability) and heterogeneous (where nodes may have varying computation capability) wireless sensor networks. We state the properties of puzzles that can be used to mitigate Sybil attack and validate using implementation and realizations in simulated environments that computational puzzles can be used to mitigate Sybil attack in wireless sensor networks.

A Cooperative Approach to Mitigating Sybil Attack in Wireless Sensor Networks

by
Gaurang Sanjeev Sinkar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2012

APPROVED BY:

Dr. David Thuente	Dr. Ting Yu

Dr. Rudra Dutta
Chair of Advisory Committee

DEDICATION

To my parents and my grandparents.

BIOGRAPHY

Gaurang was born on Feb 9, 1986 in Ratnagiri, India. He completed his bachelors degree in Electronics and Telecommunication Engineering from University of Mumbai. After completing college in 2007, he worked as a Software Engineer at Tata Consultancy Services (TCS), Bangalore. He then joined North Carolina State University in Fall 2009 to pursue Master of Science (MS) degree in Computer Science. He worked as an Intern with Research in Motion from Jan 2011 to May 2011. He would be joining Cisco Systems after his Master's program.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Rudra Dutta for showing faith in me and providing valuable guidance and insights throughout the thesis. I really admire him as an advisor, instructor, researcher and a person and would always look up to him. I would also like to thank Dr. Ting Yu and Dr. David Thuente for agreeing to be on my committee.

I would like to thank all the faculty in NC State university for conducting such good courses and providing their valuable insights.

I would like to thank my parents for encouraging me to pursue my Master's and for their constant support and encouragement during my academic career. They have been a great support throughout my life and I would always look up to them as idols. I would also like to thank Sonal Tawade, Kishor Kharbas and my other friends for their constant support. I would also like to thank Parth Pathak for his advice on various topics related to research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Sensor network Background and Applications

Sensor networks have been widely researched in recent years as they provide an economical solution to a wide range of applications. A wireless sensor network is composed of a number of low cost, battery constrained sensor nodes which communicate over a short range using wireless medium. These nodes can be rapidly deployed in places that are inaccessible to humans, which make sensors useful for a wide range of applications. A sensor network can be used for a wide variety of applications like traffic monitoring, pollution sensing, temperature and humidity sensing, sensing soil make up, building safety monitoring, monitoring the stress attached with the object, presence or absence of certain kind of object and its speed in a battle field etc. The sensor nodes are generally left unattended in a monitoring environment. These nodes sense the data they are supposed to and report this data to a base station either directly or in a multihop fashion. The base station is usually a much powerful (and potentially more secure) node and is connected to the outside world.

As the sensor nodes are left unattended in a monitoring environment, they are subject to variety of attacks. An attacker can physically compromise a sensor node and gain access to all its cryptographic material and then launch a wide range of attacks to disrupt the normal operation of a sensor network. So securing a sensor network becomes vital to achieve correct and expected behavior. However, security in sensor networks is complicated by the broadcast nature of the wireless communication and also due to lack of tamper resistant hardware used for sensor nodes (in order to reduce cost).

1.2 Sybil Attack Background and Classification

One of the attacks that an attacker can launch is a Sybil attack. In a Sybil attack, a compromised malicious node assumes identity of multiple nodes. The additional identities of the

malicious node are called Sybil nodes. The name Sybil is derived from the book Sybil, which is based on the case study of a woman with multiple personality disorder. The name was suggested by Brian Zill at Microsoft Research [1].

Sybil attack was first discovered in context of peer to peer networks in which a malicious node creates multiple identities to subvert a reputation based system. The vulnerability of such a reputation system to Sybil attack depends on how cheaply Sybil identities can be constructed and also on how the system accepts inputs from these identities [1]. In case of a sensor network, an attacker can physically compromise a sensor node and inject malicious code on the sensor node so that the malicious node assumes identities of multiple nodes (some of which may be real nodes that are part of the network, which a malicious node masquerades and some of them may be nonexistent bogus nodes).

Sybil attacks are classified according to various criteria in [12] as follows:

1. **Direct Communication vs. Indirect Communication:** This classification is based on how legitimate nodes communicate with the Sybil nodes.
   Direct communication: Legitimate nodes communicate directly with the Sybil nodes.
   Indirect communication: One or more of the compromised nodes claims to be able to reach the Sybil nodes. Packets sent to a Sybil node are routed through these compromised nodes.

2. **Fabricated vs. Stolen Identities:** This classification is based on what identities Sybil nodes assume.
   Fabricated Identity: The Sybil nodes assume identity of bogus non-existent nodes.
   Stolen Identity: The Sybil nodes assume identity of legitimate nodes that are part of the network. To avoid being detected, an attacker may destroy or disable legitimate nodes after gaining access to the keying material.

3. **Simultaneous vs. Non Simultaneous:** This classification is based on how the Sybil nodes participate in the network.
   Simultaneous participation: All the Sybil identities participate in the network at once.
   Non-Simultaneous participation: Only a fraction of Sybil identities participate in the network at a time. This can be achieved by having different identities join and leave the network or by having several physical devices swap identities.

1.3 Known threats posed by Sybil Attack

1. **Multipath Routing:**
   Sensor nodes may use geographic routing to route the data to the base station. In a sensor network, data may be routed through multiple node disjoint paths (multipath

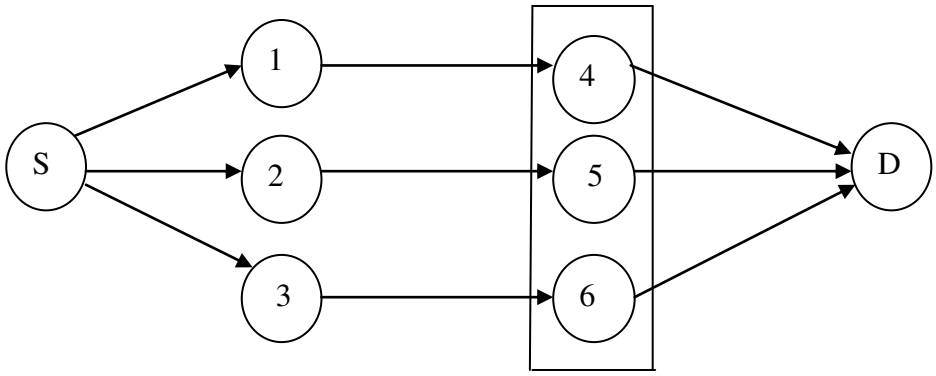


Figure 1.1: A malicious node is a part of multiple (supposed to be) node disjoint paths

routing) to achieve benefits like fault tolerance, increased bandwidth or improved security. However, a malicious node assuming multiple identities can be a part of multiple node disjoint paths which makes multipath routing ineffective. As shown in the Figure 1.1 above, nodes 4, 5 and 6 are in fact a single malicious node assuming multiple identities. A malicious node may then store the data for later crypt analysis, report the data to an attacker or randomly drop or modify the data packets. This attack has been described in [12].

2. **Data aggregation:**
   In a sensor network, in order to reduce the total number of messages sent and hence save

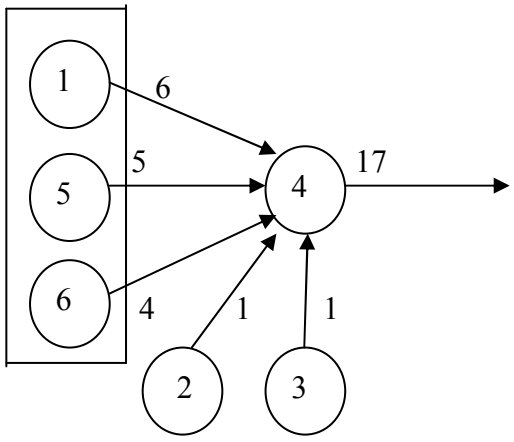


Figure 1.2: A malicious node contributing multiple times in data aggregation

energy, sensor readings from multiple nodes may be processed at aggregation points. By assuming multiple identities, a malicious node may be able to contribute to an aggregate many times. With enough Sybil nodes, an attacker may be able to completely alter an

aggregate reading. As shown in the Figure 1.2 above, the aggregate reading is altered by a malicious node assuming identities 1, 5 and 6. This attack has been described in [12].

3. **Voting based schemes:**
   Depending on the number of identities a malicious node assumes, a malicious node may

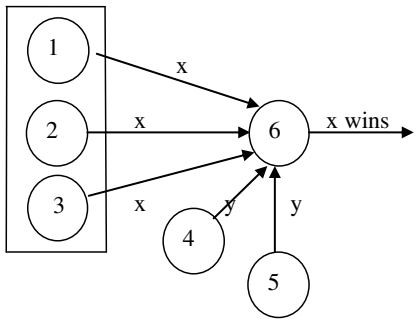


Figure 1.3: A malicious node casting multiple false votes to win over a voting process

be able to determine the outcome of any vote. A malicious node can either claim that a legitimate node is misbehaving or Sybil nodes can vouch for each other. As shown in the Figure 1.3 above, a malicious node can cast multiple votes to win over a voting based scheme. This attack has been described in [12].

4. **Fair resource allocation:**
   As shown in the Figure 1.4 below, a malicious node assuming multiple identities can obtain an unfair share of any resource. Consequently, a malicious node can cause Denial of service to legitimate nodes, and also give an attacker more resources to perform attacks. In the Figure 1.4 below, node N is responsible for granting access to resources. This attack has been described in [12].

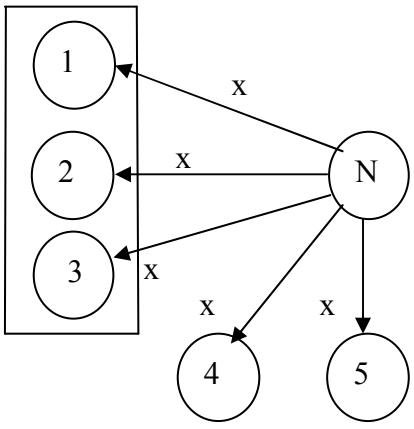


Figure 1.4: A malicious node utilizing more resources by assuming multiple identities

5. **Revoke legitimate nodes by spreading the blame:**
   A malicious node could spread the blame about legitimate nodes causing them to be revoked from the network. Even if action is taken to revoke the malicious nodes, an attacker can continue using new Sybil identities to misbehave. As shown in the Figure 1.5 below, a malicious node claims multiple identities to spread blame about an honest node 4, which may cause that node to be revoked from the network. This attack has been described in [12].

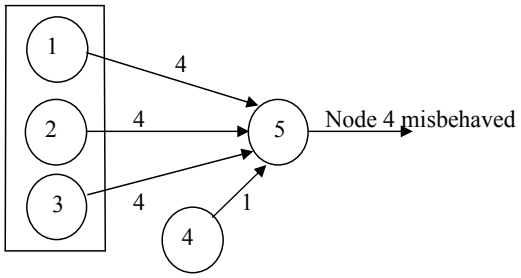


Figure 1.5: A malicious node spreading the blame about a legitimate node

6. **Distributed Storage:** A Sybil attack can defeat replicated storage and redundancy mechanisms in Peer to Peer and sensor networks. Data may be replicated accross several nodes (distributed hash table) to achieve redundancy. However due to the presence of a malicious node assuming multiple identities, data may be stored on the identities generated by same node (data may be actually stored on same node).

7. **Attack on Localization System:** In a sensor network, sensor nodes obtain their position coordinates from location signals received from multiple beacon nodes whose position coordinates are known. A malicious beacon node assuming multiple identities may report multiple incorrect location coordinates to disrupt a localization system.

1.4 Context

Consider a small portion of a wireless sensor network as shown in the Figure 1.6 below consisting of low cost sensor nodes. We propose the use of computational puzzles as a mechanism to

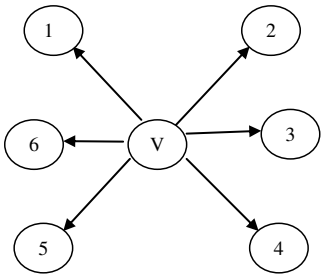


Figure 1.6: A node issuing puzzles to its neighbors

mitigate Sybil attack in Wireless Sensor networks. We define a puzzle as a cryptographic challenge that requires some computation on part of the node to whom the puzzle is issued. The intuition behind using puzzles to detect a Sybil attack is that a compromised node assuming multiple identities might not have enough computational resources to respond to all the puzzles in time. Consider a portion of the network as shown in Figure 1.6 above. When a node joins the network, it exchanges 'Hello' packets with its neighbors and establishes a neighborhood relationship. As shown, node V issues a puzzle to each of its neighbors. In Figure 1.6 above, all nodes can have same computation capability (homogeneous sensor network) or they can have different computation capability (heterogeneous sensor networks). The puzzles issued can be of same or different complexity (hardness) level. Puzzles with same complexity level require same time to be solved on the same processor. The node V which issues the puzzles is a verifier, whereas the nodes 1 to 6 who are issued a puzzle each are targets.

We list the properties of puzzles required to mitigate Sybil attack and provide examples of how existing puzzles in literature can be modified to have these properties. We show by simulation and implementation that in a homogeneous sensor network, a compromised node assuming multiple identities will not have enough computing resources to respond to all the puzzles (if the puzzles are of same or different complexity level) in time and, if a node assumes multiple identities a Sybil attack will always be detected. In case of a heterogeneous sensor network, we show that the network can detect a Sybil attack by issuing puzzles of variable complexity level and then examining the order and timing at which the puzzle solutions are received. We also suggest a bound on the number of identities a compromised node can assume in worst case, if it chooses the smartest possible way to solve the puzzles and guesses some parameters (not known to any node) correctly. We also associate a node's identity with a unique key known to that node and the base station. The base station maintains a 'Key Database' storing the keys assigned to nodes. The idea of associating a node's identity with keys has been used in [12], where every node is assigned a set of keys from a key pool and a node's identity is verified (by other nodes) by challenging the node on keys it is assigned. However in our case we assign a single key to a node and the verification is done by the base station as opposed to [12]. By associating a node's identity with a unique key, we eliminate the possibility of a Sybil attack using fabricated identities, as fabricated identities would not have valid keys.

Privacy Preservation: We believe that the Sybil detection protocol described later can be applied to detect a Sybil attack in an adhoc network of smart phones. In an adhoc network of smart phones, unique identifiers associated with phones can be used as identity certificates. But this gives away the privacy of smart phone users. As our protocol does not use the identity of nodes as a unique identifier (but just an identity key value known to the base station/service provider), the privacy of phones is preserved.

Chapter 2

Related work

2.1 Background on puzzles

We define a puzzle as a cryptographic challenge whose solution requires some computation on part of the node to whom the puzzle is issued.

[8] suggests the use of computational puzzles to mitigate Denial of service attacks. The intuition behind using puzzles to mitigate Denial of service attack is that an attacker will have too many puzzles to solve and would have to do more work in order to gain access to more resources to cause Denial of Service.

[7] uses computational puzzles to combat spam mail and to control access to a shared resource by requiring a user to compute a moderately hard puzzle.

Authentication is one of the ways to prevent Denial of Service against servers in open systems. However authentication may create new opportunities for Denial of service, since the server might have to maintain state during the authentication process and also authentication may involve expensive public key operations. [2] suggests the use of client puzzles to avoid Denial of Service during authentication. In [2], it is suggested that the client should commit its resources first during authentication and the server must verify the client before allocating the resources to the client. This would require more computational cost at the client than at the server.

[17] suggest techniques by which puzzle distribution can be outsourced to an external server (called bastion). In their work many servers rely on puzzles distributed by a single external server (bastion).

2.2 Related work on Sybil attack

The problem of Sybil attack was first identified in [5] in the context of peer-to-peer distributed systems. In [5], it is suggested that Sybil attack can be prevented by having a trusted agency certify identities. It is also suggested that in the absence of a centralized authority, a faulty (malicious) entity can always assume multiple identities and hence Sybil attack is always possible, except under extreme and unrealistic assumptions about the resources available to attackers. An entity can accept an identity by validating that identity itself (direct validation) or it can accept an identity vouched by already accepted identities. It is also proved that a faulty (malicious) entity with capability 'p' times that of minimally capable entity can present 'p' identities to any other entity. It is also suggested that an entity should accept an identity that it has directly validated simultaneously, otherwise a malicious entity can assume an arbitrarily large number of identities. In case of indirect validation, if an entity accepts an identity vouched for by 'q' identities, then a set of 'F' malicious identities can assume arbitrarily large identities, if $F \geq q$.

[9] suggests an approach based on symmetric key cryptography to defeat Sybil attack in sensor networks. In their scheme, each node shares a unique symmetric key with a base station. Nodes that wish to communicate with each other can then use a Needham-Schroeder-like protocol to verify each other and to establish a shared key (K), via the base station. The nodes can then use the key to authenticate each other and establish an encrypted link between them. The base station limits the number of neighbors a node can have to prevent a mobile node from establishing shared keys with every node in network. Thus when a node is compromised, it is restricted to only communicating with its verified neighbors. This approach might not be scalable in case of a large sensor network, as the base station can be a potential bottleneck and a single point of failure.

[20] suggests the use of lightweight identity certificates using Merkle hash tree and random key distribution. In this work, each node i in the sensor network is assigned a unique secret value (I_i). The server then computes a commitment value C_i for the node i from I_i. The server constructs a Merkle hash tree whose leaf nodes correspond to nodes in the sensor network. The label of a leaf node of the Merkle hash tree is H $(C_i \| i)$, where H$(C_i \| i)$ denotes the hash of concatenation of commitment value and identifier of node i. The server then assigns an identity certificate (using the Merkle hash tree) to the node which is stored with the node. In order to prove its identity, a node presents its certificate and then proves that it has the unique information associated with the certificate. Thus this approach uses symmetric key based mechanisms to generate identity certificates which can be used by a node to prove its identity.

[12] proposes two methods for defending against Sybil attack in sensor networks: (1) radio

resource testing, and (2) random key pre-distribution. The first approach assumes that each sensor node has only one radio and a node cannot send and receive simultaneously on more than one radio channel. Therefore by challenging a neighbor node several times on a radio communication channel which is exclusively assigned to that neighbor, a sensor node can detect Sybil nodes with a certain probability. However, how a sensor node assigns the radio channels to its neighbor nodes is a hard problem. Also a sensor node might have multiple radios through which an attacker may be able to respond to multiple challenges. The random key distribution approach associates a node's identity with keys assigned to the node from a key pool. Thus the identity of a node can be verified by challenging the node on the keys it possesses. A drawback of this approach is that if some keys are compromised, an adversary may be able to falsely claim the identities of many non-compromised sensors. This drawback has been eliminated by recent work on unique random pair wise key establishment schemes [6], [10], based on t-degree polynomials. However in this work the choice of the threshold 't' is a challenge. If 't' is too small, an attacker only needs to compromise a small percentage of sensor nodes to compromise the whole network. If 't' is too big, the storage overhead for each sensor node might be very high.

[12] also proposes the following defenses against the Sybil attack:

1. **Registration:** This defense mechanism suggests registering an identity with a centralized authority. A central authority will maintain a list of known-good identities and this list can be used to validate the identity of a legitimate node. However this list needs to be protected from being maliciously manipulated by an attacker. Also the initial deployment information that is checked against must be securely maintained by the centralized authority.

2. **Position verification:** In this defense mechanism, it is assumed that the sensor nodes are immobile after deployment. The physical position of a node is verified by the network and a Sybil attack is detected when multiple nodes appear to be in exactly same position. However securely verifying a node's position is an open problem. Also a mobile node can assume multiple identities by getting verified at one location and then move to a different location to get itself verified as a different identity. To defeat this approach all the nodes could be verified at the same time or nodes within a certain range can be verified at the same time (depending on the bound on maximum speed of the mobile attacker).

3. **Code attestation:** This defense mechanism suggests the use of remote code attestation. The intuition behind using code attestation is that code running on a malicious node must have been modified and should be different from the code running on an honest node. [15] proposes a technique to verify the memory contents of embedded devices. Thus any malicious changes in memory content can be verified.

Identity-based encryption scheme in [3] or any of the public key cryptography based schemes can also be used to defend against Sybil attack. However these schemes rely on asymmetric key cryptography. They require high computation and memory which might not be suitable for sensor nodes [13].

[4] provides a solution to detect a Sybil attack in wireless sensor networks using RSSI measurements. In this approach, it is suggested that if the ratio of RSSI at multiple monitor nodes (A, B, C and D in Figure 2.1 below) is same for 2 nodes (say S1 and S2), then the protocol can conclude a Sybil attack. The idea is explained below using an illustration mentioned in paper. Suppose that a malicious node (S in Figure 2.1 below) assumes 2 identities S1 and

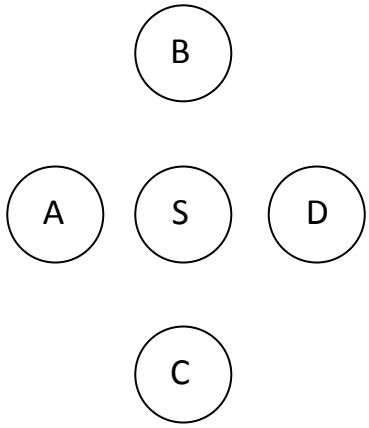


Figure 2.1: Sybil attack detection using RSSI

S2. Suppose that at a particular time instant 't1', a malicious node assumes an identity S1. Monitoring nodes A, B, C and D record the RSSI from the messages they receive from S1. Suppose that nodes B, C and D report the RSSI measurements to node A. Consider the notation where R_A^{S1} denotes the RSSI received at node A from node S. Node A computes the ratios R_A^{S1}/R_B^{S1} , R_A^{S1}/R_C^{S1} and R_A^{S1}/R_D^{S1}. Suppose that a particular time instant 't2', a malicious node assumes an identity S2. Monitoring nodes A, B, C and D record the RSSI from the messages they receive from S2. Suppose that nodes B, C and D report the RSSI measurements to node A. Node A computes the ratios R_A^{S2}/R_B^{S2} , R_A^{S2}/R_C^{S2} and R_A^{S2}/R_D^{S2} . [4] detects a Sybil attack if, $R_A^{S1}/R_B^{S1} = R_A^{S2}/R_B^{S2}$ and $R_A^{S1}/R_C^{S1} = R_A^{S2}/R_C^{S2}$ and $R_A^{S1}/R_D^{S1} = R_A^{S2}/R_D^{S2}$

[21] presents a privacy preserving protocol to detect Sybil attack in VANETs. The entities involves in this architecture are DMV (Department of Motor vehicle), RSBs (Road side boxes) and vehicles. The RSBs are wireless access points acting as intermediaries to DMV. In the scheme presented in this paper, DMV (Department of Motor vehicles) acts as a trusted party that maintains the record of vehicles. The DMV provides the vehicles with a unique pool of pseudonyms. The vehicles multiplex between pseudonyms to hide their identity. In order to

prevent the vehicles from misusing the pseudonyms to launch a Sybil attack, the pseudonyms allocated to a vehicle are hashed to a value and this value is stored at RSB (Road side boxes) and with the DMV. The RSB calculates the hashed values of overheard pseudonyms to determine if the pseudonyms came from the same pool. If the RSB suspects that the pseudonyms came from the same pool, it sends the suspected pseudonyms and the hash value to the DMV. The DMV then checks this information to detect Sybil attack. In this work RSBs are semi-trusted parties and can be compromised by attackers. RSB compromise can be detected by the DMV and a particular RSB can then be revoked.

[18], [16] detect Sybil attacks in wireless sensor network based on RSSI measurements. [11] proposes a RSSI based technique to detect Sybil nodes performing power control.

[14] uses computational puzzles as an admission control mechanism in structured peer to peer systems. The admission control system in this scheme is tree-based, with a trusted bootstrap node located at the root that admits a user into the system. The bootstrap node may be a dedicated server and need not be a peer. A node (P) that wishes to join the peer to peer network must contact the bootstrap node (B). The bootstrap node will then direct node P to a leaf node (X) already in the system (tree). The protocol works as follows: The joining node (P) has to solve the puzzle issued by X, after which node X will issue it a token. The node P has to then contact the parent of node X (X_{i-1}) and solve the puzzle issued by X_{i-1} and so on. Thus a node that wishes to be a part of the system needs to solve puzzles from all the nodes along the path from leaf to the bootstrap node. In this scheme the upper layers of the tree should be trustworthy. In order to accumulate a large number of identities to disrupt the operation of network, an attacker has to spend a long enough time (in order of days). In order to prevent an attacker from slowly accumulating identities over a long period of time, the protocol proposes a cut-off window after which every node has to readmit itself into the system. So even if an attacker patiently accumulates a large number of identities over a period of time, it has to readmit itself after the cut-off window. The disadvantage of the cut-off window is that even the legitimate nodes have to readmit themselves into the system.

[19] presents a protocol to defend against Sybil attack based on a social network among the identities. In this work, an identity is modelled as a node and trust relationships are the edges in the graph. The intuition presented in this paper is that a malicious node can create a large number of identities but only a few trust relationships. Thus a disproportionate cut exists in the graph between the Sybil nodes and good nodes, as a result of which removal of a few edges disconnects a large number of nodes from the network. This can be used to detect Sybil attack.

Chapter 3

Puzzle properties, Puzzle construction and analysis

3.1 Introduction to Puzzles:

We define a puzzle as a cryptographic challenge whose solution (response) requires some computation on part of the node to whom the puzzle is issued. Puzzles have been used previously to mitigate Denial of service attack against servers. The intuition behind using puzzles to mitigate Denial of service attack is that the requestor of a service should do some work in order to gain access to the service/resource from the server.
In this chapter, we use the term *verifier* to denote a node that is issuing puzzles, and the term *target* to denote a node to whom a puzzle is issued.
Puzzle Example:

1. Generate an 'n' bit random number 'R' and set its last 'k' bits to 0 to generate 'R1'.
2. Generate another random number 't' and hash R 't' times (as shown in Figure 3.1 below) to generate R2. The hash function can be any of the standard one-way cryptographic hash like MD5 or SHA-1.
3. Send 't', 'R1' and 'R2' to the target. The target has to return back the number 'R'. The target has to do a brute force computation starting from 'R1' upto 'R' in order to get the puzzle solution.

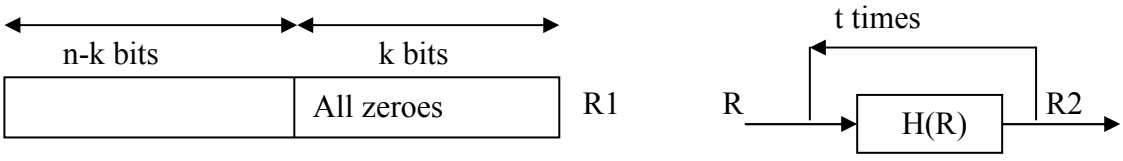


Figure 3.1: A puzzle example

Terminology:
Puzzle generator (G): A puzzle generator is a node that is responsible for generation of puzzles. If puzzle generation is computationally costly, puzzle generation can be outsourced to an external server (G). If puzzle generation is relatively less computationally costly, a verifier can be the generator of a puzzle.
Puzzle complexity level: Puzzle complexity level is proportional to the time required to solve the puzzle. A puzzle with higher complexity level will require more computation time as compared to a puzzle with lower complexity level (on the same processor).

3.2 Puzzle properties:

We list the properties that puzzles require in order to be successfully used in the approaches we provide later in this thesis.

1. Puzzles should be easy to construct and puzzle solutions should be easy to verify. But puzzles should be difficult to solve.

2. The cost of solving a puzzle (puzzle complexity level) can be tuned by the puzzle generator.

3. Puzzles are ***monolithic***. The complexity level of a puzzle cannot be estimated by looking at the puzzle. The complexity level of a puzzle is best estimated after solving the puzzle.
   Example of a non-monolithic puzzle: Let B^C denote B multiplied by itself, C times.

   (a) Generate a random number 'R'.

   (b) Generate a base 'x' by randomly selecting a number between 2 and 100. The base 'x' should be high enough to avoid precomputing the solutions to a puzzle.

   (c) Let R1 = x^R. Give 'R1' and 'x' to the target. The target has to return back the random number 'R'.

   (d) The target has to do a brute force computation starting from '0' upto 'R' (as exponent to base 'x') to solve the puzzle. The basic idea is to use the hardness of the discrete logarithm problem. In this puzzle, the target can estimate the hardness of a puzzle by looking at the puzzle since a puzzle with larger value of 'R1' is harder as compared to a puzzle with a smaller value of 'R1'. This puzzle can be converted into a monolithic puzzle as shown in example below,

   Example of a monolithic puzzle: Let H (A) denote one way cryptographic hash of number A and let B^C denote B multiplied by itself, C times. The hash function can be any standard cryptographic hash like MD5 or SHA-1.

(a) Generate a random number 'R'.

(b) Generate a base 'x' by randomly selecting a number between 2 and 100. The base 'x' should be high enough to avoid precomputing the solutions to a puzzle.

(c) Let R1 = x^R. Generate R2 = H (R1) and give 'R2' and 'x' to the target. The target has to return back the random number 'R'.

(d) The target has to do a brute force computation starting from '0' upto 'R' (as exponent to 'x') to solve the puzzle. If we did not hash 'R1' to generate 'R2', by looking at 'R1' the attacker may tell whether a puzzle is hard or easy. (If 'R1' is large the puzzle is hard and if 'R1' is small the puzzle is easier. Also seeing a large 'R1', an attacker may skip lower values of exponent or apply an algorithm similar to binary search). Hashing 'R1' to generate 'R2' does not give an attacker any clue about the puzzle difficulty, which makes the puzzle monolithic.

4. It should be computationally infeasible to precompute the solution to a puzzle.

5. A node issuing puzzles should maintain minimal details of the puzzle in order to verify the solution.

6. Having a solution to one puzzle should not make solving any other puzzle easier.

7. An individual puzzle should preferably not be easily parallelizable. A puzzle can be solved in parallel if, it can be divided into parts and each part can be solved independently of each of the other part. Puzzles which solely rely on brute force computation can be solved in parallel.

8. The puzzles should not consume a lot of network bandwidth (the number of bytes used to represent a puzzle construction and solution should be as minimum as possible).

9. Puzzles should be lightweight. Puzzle construction and solution should not consume a lot of sensor battery.

3.3 Puzzle generation algorithms and Analysis:

We categorize puzzles into 3 types and list the properties and examples of each type of puzzle. We assume that for all the puzzles, to do a brute force computation, a node starts from all 0's and increments by 1 each time.

Type 1 Puzzle properties: This type of puzzles are easy to generate and puzzle solutions are easy to verify, but the puzzles are difficult to solve. Hence for this type of puzzles, a verifier may be the generator of puzzles or each node can have a precomputed library of puzzles, if storage

permits. Puzzle complexity (hardness) level can be tuned by the puzzle generator. Puzzles of a particular complexity level i would require time T_i to be solved in best case and time T_{i+1} to be solved in worst case (where $T_i < T_{i+1}$, for all i). Thus the worst case time for a puzzle of complexity level i (T_{i+1}) is the best case time for a puzzle of complexity level $i + 1$. So while analyzing the response time of puzzle solution during Sybil detection, average time required to solve the puzzle should be considered. These puzzles rely solely on brute force computations and hence an individual puzzle can be parallelized. Also it is computationally infeasible to precompute the solutions to this type of puzzle and knowing the solution of one puzzle does not make solving any other puzzle easier. The puzzles are monolithic, since given a puzzle it is difficult to guess the complexity level of a puzzle.

Example 1:

1. Generate an n-bit random number 'R1' with its last 'k' bits set to 0 (n > k) as shown in Figure 3.2 below.

Figure 3.2: Type 1 puzzle: Example 1 step 1

2. Divide 'k' bits into 'm' blocks. So each block has 'k/m' bits as shown in Figure 3.3 below.

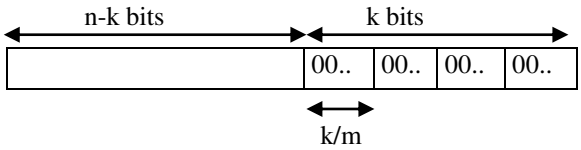


Figure 3.3: Type 1 puzzle: Example 1 step 2

3. To generate a puzzle of least complexity level (complexity level 0), generate a random number between 1 and $2^{k/m}$. To generate a puzzle of complexity level 1, generate a random number between $2^{k/m}$ and $2^{2k/m}$. Thus, to generate a puzzle of complexity level 'c' (where 0 <= c <= m-1), generate a random number (R2) between $2^{ck/m}$ and $2^{(c+1)k/m}$. Do a bit wise OR of this random number and R1. Call the number generated as a result of this as R3.

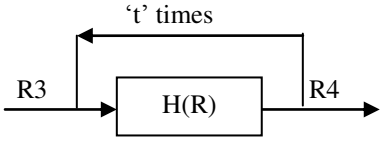


Figure 3.4: Type 1 puzzle: Example 1 step 3

4. Randomly generate a number 't'. Hash R3 't' times to generate a number R4 as shown in

Figure 3.4 above. The hash function can be any of the standard one-way cryptographic hash like MD5 or SHA-1.

5. Give R1, R4 and 't' to the target and the target has to return back 'R3'. The only way to generate R3 is by brute force computation starting from 'R1' upto 'R3'. The complexity level (hardness) in puzzle example above is controlled by the parameters 'k', 'm' and 't'.

Computation cost: The computation cost for the generator involves generating the random number (R1) with last 'k' bits set to 0. Also the generator has to invoke the hash function 't' times. The target however has to do R2 brute force attempts and thus has to invoke the hash function t*R2 times.

Puzzle analysis:

1. The complexity level of the puzzle can be increased or decreased exponentially by increasing or decreasing the value of 'k'. This puzzle generation algorithm divides the last 'k' bits into 'm' blocks. The generator has to generate a random number within these blocks in order to generate puzzles of specific complexity levels. By generating a random number within these blocks, the generator has more fine grained understanding of complexity level of the puzzle. Also the puzzle generator can do more fine grained tuning of the complexity level of puzzle.

2. Since we generate a new random number (NONCE) R1 every time, it is computationally infeasible to precompute the solution to a puzzle and knowing the solution to a particular puzzle instance does not make the solution to any other instance easy.

3. The verifier just has to remember the random numbers 'R1', 'R4', and 't' that it gave to the target and the solution 'R3'. So the verifier does not have to maintain a lot of state about the puzzle in order to verify the solution.

4. Since this puzzle relies solely on brute force attempts to compute the solution to the puzzle, this puzzle can be parallelized.

Example 2: Let H (A) denote one way cryptographic hash of number A and let B^C denote B multiplied by itself, C times. The hash function can be any standard cryptographic hash like MD5 or SHA-1.

1. Generate a random number 'R'.

2. Generate a base 'x' by randomly selecting a number between 2 and 100. The base 'x' should be high enough to avoid precomputing the solution to a puzzle.

3. Let R1 = x^R. Generate R2 = H (R1) and give 'R2' and 'x' to the target. The target has to return back the random number 'R' to the verifier.

4. The target has to do a brute force computation starting from 0 upto 'R' (as exponent to 'x') to solve the puzzle. The basic idea is to use the hardness of the discrete logarithm problem. If we did not hash 'R1' to generate 'R2', by looking at 'R1' an attacker may tell whether a puzzle is hard or easy. (If 'R1' is large the puzzle is hard and if 'R1' is small the puzzle is easier. Also seeing a large 'R1' an attacker may skip lower values of exponent or apply an algorithm similar to binary search). Hashing 'R1' to generate 'R2' does not give an attacker clue about the puzzle difficulty, which makes the puzzle monolithic.

Computation cost: The generator has to do O (log_2 (R)) multiplications and one hash operation to generate the puzzle. However to solve the puzzle, the target has to do 'R' multiplications, 'R' hash operations and 'R' comparison operations.

Puzzle analysis:

1. The complexity level of puzzles can be tuned by varying the values of 'x' and 'R'. Puzzles with higher value of 'x' are harder than the puzzles with lower value of 'x'. For a particular value of 'x', the complexity level of a puzzle can be increased by generating a larger random number 'R'.

2. Since we generate a new base 'x' and random number (NONCE) R every time, it is computationally infeasible to precompute the solution to a puzzle and knowing the solution of a particular puzzle instance does not make the solution to any other instance easy.

3. The verifier just has to remember the numbers 'x' and 'R2' that it gave to the target and also 'R'. Thus the verifier does not have to maintain a lot of state about the puzzle.

4. Since this puzzle relies only on brute force computation to get the solution, this puzzle can be parallelized.

Type 2 Puzzle properties: For this type of puzzles, puzzle generation requires almost same time as solving the puzzle since the generator has to solve the puzzle first in order to generate the puzzle as shown in example 3 below. So puzzle generation may be outsourced to an external server. The generator of the puzzle can estimate the exact time required to solve the puzzle (no average case analysis required) and hence has more fine grained understanding of puzzle complexity. Puzzle complexity level can be tuned by the puzzle generator. These puzzles cannot be parallelized since the input to a particular round depends on the output of previous round (as shown in example 3 below). Also it is computationally infeasible to precompute the solutions to this type of puzzle and knowing the solution of one puzzle does not make any other puzzle easier. The puzzles are monolithic, since given a puzzle it is difficult to guess the complexity level of the puzzle.

Example 3:

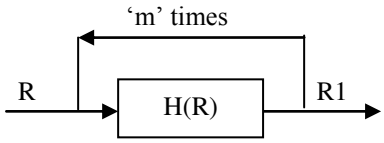


Figure 3.5: Type 2 puzzle example

1. Generate random numbers 'm' and 'R'. Hash 'R' 'm' times as shown in the Figure 3.5 above to generate 'R1'. The hash function can be any standard one-way cryptographic hash like MD5 or SHA-1.

2. Give 'R' and 'R1' to target. The target should return 'm' and $H_{(m-1)}(R)$. The complexity level (hardness) in puzzle example 3 is controlled by the parameter 'm'. The task of generating such puzzles and puzzle solutions may be offloaded to an external trusted server since the generator of the puzzle has to do almost equivalent work as that of the target. We do not want the verifier to spend a lot of its computation time in puzzle generation.

Computation cost: The generator has to generate random numbers 'm' and 'R' and hash 'R' 'm' times as shown in the Figure 3.5 above. So the generator requires 'm' hash operations to generate the puzzle. The target also requires 'm' hash operations to solve the puzzle. To add asymmetry in computation, we may generate R2 = H (R1) and give 'R' and 'R2' to the target (instead of R and R1). In this case, the generator has to do 'm' hash operations whereas the target has to do '2m' hash operations.

Puzzle Analysis:

1. The complexity level of the puzzle can be increased or decreased by increasing or decreasing the value of 'm' respectively. Since the generator and the target do same amount of computation to generate and solve the puzzle respectively, the generator can estimate the exact amount of time required to solve the puzzle. Hence the complexity of puzzle can be tuned to a finest level.

2. Since the generator generates a new random number (NONCE) R every time, it is computationally infeasible to precompute the solutions to a puzzle and knowing the solution to a particular puzzle instance does not make the solution to any other instance easy.

3. The verifier just has to remember the 'R' and 'R1' that it gave to the target and also the 'm' and $H_{(m-1)}(R)$. Thus the verifier does not have to maintain a lot of state about the puzzle.

4. This puzzle cannot be parallelized since to generate the output of a particular round, the output of previous round is required as input.

Example 4:

1. Generate a random number 'a' and another random number 'm'. Set R = a
2. Square 'a' and add 1 to it. $a = a^2 + 1$
3. Repeat step 2 'm' times.
4. Send 'R' and H (a) to the target. The target has to return back 'm'.

Computation cost: In order to generate a puzzle, the generator has to do 'm' squaring operations and 1 hash operation. To solve this puzzle, the target has to do 'm' squaring operations and 'm' hash operations.

1. The complexity level of the puzzle can be increased or decreased by increasing or decreasing 'm' respectively. The generator can estimate the exact time required to solve the puzzle since both the generator and target do the same operations to generate and solve the puzzle respectively.
2. Since the generator generates new random numbers 'a' (NONCE) and 'm' every time, it is computationally infeasible to precompute the solutions to a puzzle and knowing the solution of a particular puzzle instance does not make the solution to any other instance easy.
3. The verifier just has to remember the value 'R' that it gave to the target and also the value 'm'. Thus the verifier does not have to maintain a lot of state about the puzzle.
4. This puzzle cannot be parallelized since the output of one step acts as an input to the next step.

Type 3 Puzzle: For this type of puzzles, puzzle generation is harder than Type 1 puzzle but is easier than Type 2 puzzle. Hence for this type of puzzles, a verifier may be the generator of puzzles or each node can have a precomputed library of puzzles, if storage permits. Puzzle complexity (hardness) level can be tuned by the puzzle generator. Puzzles of a particular complexity level i would require time T_i to be solved in best case and T_{i+1} to be solved in worst case (where $T_i < T_{i+1}$, for all i). Thus the worst case time for a puzzle of complexity level i (T_{i+1}) is the best case time for a puzzle of complexity level $i + 1$. So while analyzing the response time of puzzle solution during Sybil detection, average time required to solve the puzzle should be considered. These puzzles cannot be completely parallelized (as explained in example 5 below). Also it is computationally infeasible to precompute the solution to this type of puzzle and knowing the solution of one puzzle does not make any other puzzle easier. The puzzles are monolithic, since given a puzzle it is difficult to guess the complexity level of the

puzzle (as explained in example 5 below).

Example 5: Consider the tree in Figure 3.6 below which has the following property. The right

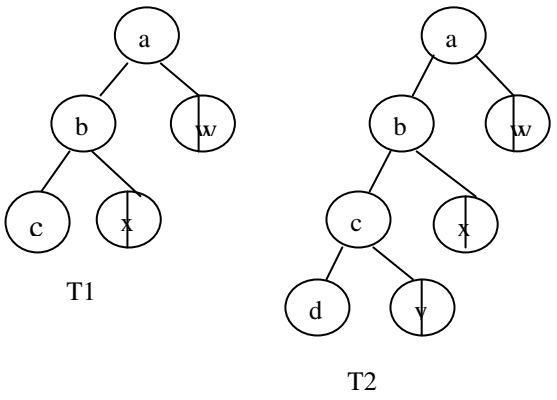


Figure 3.6: Type 3 puzzle example

child of every node is a leaf. There is only one node whose left child is a leaf. This tree may be considered a special case of a height biased leftist tree. The tree above can be used to construct a puzzle that cannot be completely parallelized. Let H (A) denote one way cryptographic hash of number A. The hash function can be any standard one-way cryptographic hash like MD5 or SHA-1. Let A ‖ B denote the concatenation of A and B. A node half cut symbolizes that the last 'k' bits are set to 0 and partial values (w1, x1, y1) are given to the target.

Puzzle generation algorithm:

1. Consider the figure 3.6 (T2) above. The puzzle generator randomly generates w, x, y, d and Initialization vector (IV) and then does the following computation.
   c = H (d ‖ H (y ‖ IV)) ..... (1)
   b = H (c ‖ H (x ‖ y)) ..... (2)
   a = H (b ‖ H (w ‖ x)) ..... (3)
   p = H (a ‖ w ‖ x ‖ y) ..... (4)

2. Set the last 'k' bits of w, x and y to 0 and generate w1, x1 and y1.

3. The target is given a, b, c, d, w1, x1 and y1. The target has to return p.

Puzzle solution algorithm:

1. Given d, c and IV, substitute in (1) and do a brute force starting from y1 to get y.

2. Given c, b and y, substitute in (2) and do a brute force starting from x1 to get x.

3. Given b, a and x, substitute in (3) and do a brute force starting from w1 to get w.

4. Return p = H (a ‖ w ‖ x ‖ y). The verifier just needs to store the value of 'p' and the identifier of target to which the puzzle was sent. The only portion that can be parallelized is the brute force operation (for example to generate y from y1 etc).

Puzzle analysis:

1. The complexity level of the puzzle can be controlled using two parameters: 1). The depth of the tree. 2). The number of brute force attempts required to solve the puzzle (by adjusting the parameter k).

2. Also since the puzzle parameters are generated randomly, it is computationally infeasible to precompute the solution to a puzzle and knowing the solution of a particular puzzle instance does not make the solution to any other instance easy.

3. The verifier just has to remember the value 'p' (can be generated from a, w, x and y). So the verifier does not have to maintain a lot of state about the puzzle.

4. The only portion of the puzzle that can be parallelized is the brute force operation. Thus the puzzle can only be partially parallelized. The puzzles which can be completely parallelized have less construction cost whereas the puzzles which cannot be parallelized have high construction cost. This puzzle is a tradeoff between the two.

5. The puzzles are monolithic since a puzzle with a longer tree depth does not necessarily guarantee a more difficult puzzle as compared to a puzzle with shorter tree depth (as explained in example 6 below).

Example 6:
Consider 2 trees (T1 and T2) referring to Figure 3.6 above. T1 has a height of 2 and has following variable values, w = 2650, x = 1500, w1 = 0, x1 = 0. T2 has a height of 3 and has following variable values, w = 495, x = 650, y = 560, w1 = x1 = y1 = 0. The operations used in puzzle computation are hash and concatenation. Suppose that a hash operation costs 5 units of work and a concatenation operation costs 1 unit of work. So to calculate variables 'c', 'b' and 'a', it takes 12 units of work for each brute attempt. For tree T1, calculating 'c' requires 560 brute force attempts, calculating 'b' requires 650 brute force attempts and calculating 'a' requires 495 attempts which amounts to a total of 20460 units of work (560*12 + 650*12 + 495*12). For tree T2, calculating 'b' takes 1500 brute force attempts and calculating 'a' takes 2650 brute force attempts which amounts to total of 49800 units of work (1500*12 + 2650*12). Thus puzzle tree of shorter height can be more difficult to solve as compared to a tree of greater height. This is also true for different pairs of hash and concatenation operation costs.

Chapter 4

Homogeneous Sensor Network

We define a homogeneous sensor network as a sensor network consisting of nodes with similar computation capability.
Problem Statement: Consider a homogeneous wireless sensor network consisting of a large number of nodes with similar computation capability. The goal is to mitigate Sybil attack in such a sensor network.

4.1 Assumptions

We make a set of assumptions; our assumptions largely serve to make the problem conditions specific, and are not restrictive.

The sensor nodes are monitored by a base station. Once deployed, each node is static or the degree of mobility is very less. We assume that losses in the network are minimal. We also assume a reliable upper bound on the time taken by a node with slowest computational speed (processor) to solve the simplest of puzzles. This bound information is available to the base station but is not available to any sensor node. We assume that the base station is in a well-protected area and cannot be compromised. This is a valid assumption in case of critical applications like battlefield environments. We also assume that the packets are authenticated and any tampering with the packet data can be detected. We assume that majority of nodes in the network are uncompromised. We assume that all the sensor nodes have a single processor (a sensor node does not have parallel processing capability). This is a valid assumption in case of sensor nodes where maintaining low cost is one of the major goals. We assume that every node has a 2 secret values (a unique identity key and symmetric encryption key), which are known only to that node and the base station. An identity key is used as a unique identifier for the node whereas encryption key is used to encrypt data that needs to be understood only by the base station. An attacker can physically compromise legitimate sensor nodes to read off

the keys and then use these nodes to attack the network.

4.2 Detection in homogeneous sensor networks

Consider a homogeneous wireless sensor network consisting of a large number of nodes with similar computation capability. A verifier node will issue a puzzle of same complexity level to each of its neighbors (target node) and then report the timing and order in which the puzzle solutions are received to the base station. The base station will then examine this information to detect the presence of a Sybil node.

4.3 Sybil detection algorithm

A verifier node issues puzzles of same complexity level to all of its neighboring targets and reports the timing and order in which the puzzle solutions are received to the base station. If a target node does not respond with puzzle solution in time (as explained in Heuristic 1 later), the reputation of that target node is reduced. The base station can use any of the standard reputation systems to keep track of the reputation of the nodes. If the reputation of a node falls below a threshold, the network should take appropriate action against that target node. Consider the Figure 4.1 below in which a verifier intends to verify the identity of multiple target

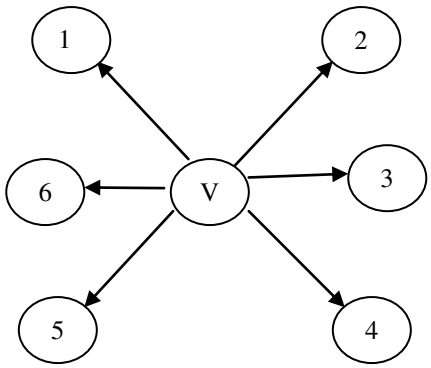


Figure 4.1: A verifier node issuing puzzles to targets in homogeneous sensor network

nodes. These target nodes may be just a single node or a compromised node assuming multiple identities.
The protocol works as follows:

1. Suppose that a verifier has 'k' neighbors (targets). The verifier issues 'k' different puzzles of same complexity level such that each of the target node gets a puzzle. We assume a reliable upper bound on the amount of time required by the node with slowest computation speed to solve each of the puzzle.

2. In normal circumstances, in absence of a Sybil attack, all the nodes should get back at almost the same time. If a malicious node assumes identity of multiple nodes, then the malicious node will get multiple puzzles and it won't have enough computation resources to reply with all the puzzle solutions in time. If this happens during multiple verification rounds, the network can conclude that there is a Sybil attack. As the conclusion is based on multiple verification rounds, false positives that occur because of packet losses can be reduced. Following is the pseudo-code of the detection algorithm at the verifier sensor node and the base station.

Pseudocode at verifier sensor node:

1. *for (i = 1 to k)* {
   if (Verifier is the generator of puzzles) {
   P_i = *Generate puzzle of type 1*
   } *else* {
   P_i = *Puzzle of type 2 from external server*
   }
   }
   for (i = 1 to k) {
   Send puzzle P_i *to neighbor* N_i
   }

2. *Report the order and timing at which the puzzle solutions are received to the base station*

Psuedocode at base station:

1. *if (Heuristic 1 fails)*
   Sybil attack is possible. Reduce the reputation of node N_i

2. *if (Solution to puzzle is incorrect)*
   Sybil attack is possible. Reduce the reputation of node N_i

3. *If (Reputation of Node* N_i *reduces below a threshold)*
   Sybil attack detected. Take appropriate action against the node.

In step 1 of the pseudo-code at the base station, the base station applies following Heuristic to detect the presence of a Sybil node.

Heuristic 1:

An honest node will always respond with puzzle solutions in time. A malicious node assuming multiple identities will not have enough computing resources to respond to all the puzzles in

time. The base station compares the expected time of puzzle solution with the actual time at which the puzzle solution is received to detect a Sybil attack.

The table below summarizes the approaches that can be followed by a malicious node (assuming multiple identities) to respond to puzzles. In each case a Sybil attack is detected. These approaches are explained in detail in the following discussion.

Table 4.1: Table summarizing various approaches that could be followed by a malicious node to solve puzzles

Approach by a compromised node	**Defense mechanism**	**Sybil Attack Detected?**
A compromised node assuming multiple identities solves puzzles sequentially. Replies with all puzzle solutions at once.	Heuristic 1	Yes
A compromised node assuming multiple identities solves puzzles sequentially. Replies to a puzzle as soon as it gets the solution to a puzzle.	Heuristic 1	Yes
A compromised node assuming multiple identities solves puzzles in parallel. Replies with all puzzle solutions at once.	Heuristic 1	Yes
A compromised node assuming multiple identities solves puzzles in parallel. Replies to a puzzle as soon as it gets the solution to a puzzle.	Heuristic 1	Yes

Approach 1: A compromised node assuming multiple identities solves the puzzles sequentially and replies to all the puzzles at once after completely solving all the puzzles.

Defense against approach 1: Heuristic 1

Example:

Suppose that a verifier issues puzzles of type 2 (of same complexity level) and the time required to solve each puzzle is 'T' units. Suppose that a malicious node 'a' assumes identity of nodes 'a', 'b' and 'c'. So node 'a' will receive 3 puzzles P1, P2 and P3 of same complexity (which the verifier has issued to nodes 'a', 'b' and 'c' respectively). Node 'a' can solve the puzzles P1,

P2 and P3 sequentially in 6 possible ways (orders). Table 4.2 below shows that in whatever order the malicious node chooses to solve the puzzles, it will not be able to respond to all of the puzzles in time. Thus a Sybil attack will be detected by Heuristic 1.

Table 4.2: Table illustrating possibilities in Approach 1

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack Detected?
P1 P2 P3	T	3T	T	3T	T	3T	Yes
P1 P3 P2	T	3T	T	3T	T	3T	Yes
P2 P1 P3	T	3T	T	3T	T	3T	Yes
P2 P3 P1	T	3T	T	3T	T	3T	Yes
P3 P1 P2	T	3T	T	3T	T	3T	Yes
P3 P2 P1	T	3T	T	3T	T	3T	Yes

Approach 2: A compromised node assuming multiple identities solves the puzzles sequentially and replies to a puzzle as soon as it solves the puzzle.

Defense against approach 2: Heuristic 1

Example:

Suppose that a verifier issues puzzles of type 2 (of same complexity level) and the time required to solve each puzzle is 'T' units. Suppose that a malicious node 'a' assumes identity of nodes 'a', 'b' and 'c'. So node 'a' will receive 3 puzzles P1, P2 and P3 of same complexity level (which the verifier has issued to nodes 'a', 'b' and 'c' respectively). Node 'a' can solve the puzzles P1, P2 and P3 sequentially in 6 possible ways (orders). Table 4.3 below shows that in whatever order the malicious node chooses to solve the puzzles, it will not be able to respond to all of the puzzles in time. Thus a Sybil attack will be detected by Heuristic 1.

Table 4.3: Table illustrating possibilities in Approach 2

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack Detected?
P1 P2 P3	T	T	T	2T	T	3T	Yes
P1 P3 P2	T	T	T	3T	T	2T	Yes
P2 P1 P3	T	2T	T	T	T	3T	Yes
P2 P3 P1	T	3T	T	T	T	2T	Yes
P3 P1 P2	T	2T	T	3T	T	T	Yes
P3 P2 P1	T	3T	T	2T	T	T	Yes

Approach 3: A compromised node solves the puzzles in parallel. Suppose that a malicious node assumes identity of 'q' nodes (so it will receive 'q' puzzles). The malicious node creates a thread for each of the puzzle. The malicious node then guesses an appropriate epoch interval 'T' and solves each puzzle for T/q time during this epoch interval and then switches to another puzzle. The malicious node replies to all puzzles at once after completely solving all the puzzles.

Defense against approach 3: Heuristic 1

Example:

Suppose that a verifier issues puzzles of type 2 (of same complexity level) and the time required to solve each puzzle is 'T' units. Suppose that a malicious node 'a' assumes identity of 3 nodes 'a', 'b' and 'c'. Node 'a' will receive 3 puzzles (of type 2) P1, P2 and P3 of same complexity level (which the verifier has issued to nodes 'a', 'b' and 'c' respectively). Table 4.4 below shows that if the malicious node solves the puzzles in parallel within an epoch (in any order), and responds to all the puzzles at once after solving all the puzzles completely, it will not be able to respond to all the puzzles in time and hence a Sybil attack would be detected by Heuristic 1.

Table 4.4: Table illustrating possibilities in Approach 3

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack Detected?
P1 P2 P3	T	3T	T	3T	T	3T	Yes
P1 P3 P1	T	3T	T	3T	T	3T	Yes
P2 P1 P3	T	3T	T	3T	T	3T	Yes
P2 P3 P1	T	3T	T	3T	T	3T	Yes
P3 P1 P2	T	3T	T	3T	T	3T	Yes
P3 P2 P1	T	3T	T	3T	T	3T	Yes

Approach 4: A compromised node solves the puzzles in parallel. Suppose that a malicious node assumes identity of 'q' nodes (so it will receive 'q' puzzles). The malicious node creates a thread for each of the puzzle. The malicious node guesses an appropriate epoch interval 'T' and solves each puzzle for T/q time during this interval and then switches to another puzzle. The malicious node replies to a puzzle as soon as it solves the puzzle.

Defense against approach 4: Heuristic 1

Example:

Suppose that a verifier issues puzzles of type 2 (of same complexity level) and the time required to solve each puzzle is 'T' units. Suppose that the malicious node 'a' assumes identity of 3 nodes 'a', 'b' and 'c'. Node 'a' will receive 3 puzzles (of type 2) P1, P2 and P3 of same complexity level (which the verifier has issued to nodes 'a', 'b' and 'c' respectively). Table 4.5 below shows that if the malicious node 'a' solves the puzzles in parallel within an epoch (in any order), and responds to a puzzle as soon as it solves the puzzle, it will not be able to respond to all the puzzles in time and hence a Sybil attack would be detected by Heuristic 1.

Table 4.5: Table illustrating possibilities in Approach 4

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack Detected?
P1 P2 P3	T	2.33T	T	2.66T	T	3T	Yes
P1 P3 P2	T	2.33T	T	3T	T	2.66T	Yes
P2 P1 P3	T	2.66T	T	2.33T	T	3T	Yes
P2 P3 P1	T	3T	T	2.33T	T	2.66T	Yes
P3 P1 P2	T	2.66T	T	3T	T	2.33T	Yes
P3 P2 P1	T	3T	T	2.66T	T	2.33T	Yes

The protocol described above will work for a homogeneous sensor network where all the nodes have same computation capability. However in case of a heterogeneous sensor network, a compromised node with a higher computation capability can solve multiple puzzles in time and hence will be able to masquerade multiple nodes. Also it can be argued that an attacker can have arbitrarily high computation power. An attacker may physically compromise a sensor node and connect it via a low latency medium (like ethernet) to an external computationally powerful server. The defense against this kind of approach by an attacker is discussed in Chapter 5.

Chapter 5

Heterogeneous Sensor Network

We define a heterogeneous sensor network as a sensor network in which nodes may have dissimilar computing capabilities.

Problem Statement: Consider a heterogeneous wireless sensor network consisting of a large number of nodes which may have dissimilar computing capabilities. The goal is to mitigate Sybil attack in such a sensor network.

We extend the protocol for homogeneous sensor network to work in case of a heterogeneous sensor network using following mechanisms.

1. A verifier node issues monolithic puzzles (a puzzle whose complexity level cannot be estimated by looking at the puzzle. The complexity level of the puzzle is best estimated after solving the puzzle completely) of different complexity levels to its neighbors. We show that by examining the timing and the order in which the puzzle solutions are received, the network can detect a Sybil attack under most of scenarios.

2. Associating a node's identity with a unique identity key and verifying the identity of that node by challenging the node on that key. Every node has a unique identity key that is known to that node and to the base station. The base station maintains a Key database with a relation having Node Id, Key and the Key expiration time as attributes.

The idea of associating a node's identity with keys has been used in [12], where a node's identity is associated with a set of keys assigned to that node from a key pool. A node's identity can be proved by verifying part or all of the keys that this node claims to have. As we use a single key to identify a node's identity, we require less storage as compared to scheme in [12], however our verification scheme is centralized as opposed to scheme in [12].

5.1 Assumptions

We make a set of assumptions; our assumptions largely serve to make the problem conditions specific, and are not restrictive.

The sensor nodes are monitored by a base station. Once deployed, each node is static or the degree of mobility is very less. We assume that losses in the network are minimal. We also assume a reliable upper bound on the time taken by a node with slowest computational speed (processor) to solve the simplest of puzzles. This bound information is available to the base station but is not available to any sensor node. We assume that the base station is in a well-protected area and cannot be compromised. This is a valid assumption in case of critical applications like battlefield environments. We also assume that the packets are authenticated and any tampering with the packet data can be detected. We assume that majority of nodes in the network are uncompromised. We assume that all the sensor nodes have a single processor (a sensor node does not have parallel processing capability). This is a valid assumption in case of sensor nodes where maintaining low cost is one of the major goals. We assume that every node has a 2 secret values (a unique identity key and symmetric encryption key), which are known only to that node and the base station. An identity key is used as a unique identifier for the node whereas encryption key is used to encrypt data that needs to be understood only by the base station. An attacker can physically compromise legitimate sensor nodes to read off the keys and then use these nodes to attack the network.

5.2 Detection in heterogeneous sensor networks

Consider a heterogeneous wireless sensor network consisting of a large number of nodes with dissimilar computing capabilities. A verifier node will issue a challenge and a puzzle to each its neighbors (target node), and then report the response to the challenge and the order and timing at which the puzzle solutions are received to the base station. The base station will then examine this information to detect the presence of a Sybil node.

5.3 Sybil detection algorithm

Assume that a verifier has 'k' neighbors. A verifier issues puzzles such that each of its neighbor gets a puzzle of different complexity level. Along with the puzzle, a verifier also sends a challenge 'Z_i' to each target 'T_i'. The target has to respond with the puzzle solution along with H ($Z_i \parallel$ H (K_i)), where H denotes a one-way cryptographic hash (like MD5 or SHA-1) and K_i denotes the unique identity key of the target node 'T_i'. The verifier reports the following information to the base station,

1. The timings and order in which the puzzle solutions are received.
2. The challenge (Z_i) it sent to each of the target T_i, and the response to challenge it received from each of the target.

The base station will verify the correctness of the response to challenge Z_i (as it knows the unique identity key associated with nodes) and also examine the order and timing at which the puzzle solutions are received to detect the presence of a Sybil node. In order to prevent false detection due to malicious readings reported by a compromised verifier against a legitimate node, we put a threshold on the number of times a report from a node against a particular node should be considered.

Pseudo-code at verifier:

1. *for (i = 1 to k) {*
   if (Verifier is the generator of puzzles) {
   P_i = Generate puzzle of Type 1
   } else {
   P_i = Puzzle of type 2 from external server
   }
   }
   for (i = 1 to k) {
   Send puzzle P_i and challenge Z_i to neighbor N_i
   }
2. *Report the challenge, response to the challenge from each target, and the order and timing at which the puzzle solutions are received to the base station.*

Pseudo-code at base station:

1. *For a particular node N_i*
   If (Response to challenge Z_i is incorrect)
   Sybil attack is possible. Reduce the reputation of node N_i
2. *If (Response to puzzle P_i is incorrect)*
   Sybil attack is possible. Reduce the reputation of node N_i
3. *If (Node N_i detected suspicious from Heuristic 1 or 2)*
   Sybil attack is possible. Reduce the reputation of node N_i
4. *If (Reputation of Node N_i reduces below a threshold)*
   Sybil attack detected. Take appropriate action against node N_i.

Any standard reputation system can be used by the base station to keep track of the reputation of sensor nodes. By requiring a target to respond to a challenge (Z_i) and verifying the response at the base station, the protocol can easily detect Sybil nodes with bogus or fabricated identities.

In step 3 of pseudo-code at base station, the base station does an analysis on the order and timing at which the puzzle solutions are received to detect a Sybil attack. This analysis can be done offline or offloaded to an external dedicated server. Following heuristics can be applied at the base station to detect the presence of a Sybil node.

Heuristic 1:Timing only heuristic

An honest node will always be able to respond with puzzle solutions in time, whereas a malicious node assuming multiple identities might be late in responding with puzzle solutions. This heuristic attempts to detect an attack by comparing the expected puzzle solution time with the actual time at which the puzzle solution is received.

Example:

A node with computation speed 'p' times faster than the computation speed of slowest node in the network can assume 'p' identities. Suppose that a compromised node 'a' with computation speed 3 times of the computation speed of the slowest node in the network assumes 3 identities 'a', 'b' and 'c'. As a result, it has to respond to puzzles issued to nodes 'a', 'b' and 'c'. Assume that a verifier issues 3 puzzles (of type 2) P1, P2 and P3 to nodes 'a', 'b' and 'c' respectively that take time T, 3T and 9T to be solved by a node with slowest computation speed (S). As node 'a' is 3 times faster than S, it can solve these puzzles in time 0.33T, T and 3T respectively. If node 'a' solves these puzzles sequentially in any order other than (P1, P2, P3), it will not be able to respond with all the puzzle solutions in time as shown in Table 5.1 below. Assume that node 'a' solves the puzzles sequentially in order P3, P2, P1. It will then respond to P1, P2 and P3 in time 4.33T, 4T and 3T respectively as shown in the Table 5.1 below. Thus it is late in responding to puzzles P1 and P2 and hence Heuristic 1 will detect a Sybil attack.

Table 5.1: Table illustrating Heuristic 1

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Attack Detected?
P1 P2 P3	T	0.33T	3T	1.33T	9T	4.33T	No
P1 P3 P2	T	0.33T	3T	4.33T	9T	3.33T	Yes
P2 P1 P3	T	1.33T	3T	T	9T	4.33T	Yes
P2 P3 P1	T	4.33T	3T	T	9T	4T	Yes
P3 P1 P2	T	3.33T	3T	4.33T	9T	3T	Yes
P3 P2 P1	T	4.33T	3T	4T	9T	3T	Yes

Extension to Heuristic 1: If we estimate the processing speed of a node based on the time it takes to solve a puzzle, the estimated speed of an honest node will always be consistent. However the estimated processing speed of a malicious node assuming multiple identities might vary depending on the order in which the malicious node solves the puzzles. This heuristic extension attempts to detect a Sybil attack based on variations in the estimated speed of a node in subsequent rounds of detection. The processing speed of an honest node will always be estimated consistently by multiple verifier nodes whereas the estimated processing speed of a malicious node assuming multiple identities might vary accross multiple verifiers.

Example: Let S denote the node with slowest computation speed. Suppose that a compromised node 'a' with computation speed 3 times of the computation speed of the slowest node in the network assumes 3 identities 'a', 'b' and 'c'. As a result, it has to respond to puzzles issued to 'a', 'b' and 'c'. Assume that a verifier issues 3 puzzles (of type 2) P1, P2 and P3 of complexity level C1, C2 and C3 to nodes 'a', 'b' and 'c' respectively. Suppose that the puzzles with complexity level C1, C2 and C3 can be solved by node S in time T, 4T and 8T respectively. Node 'a' which is 3 times faster than S can solve these puzzles in time 0.33T, 1.33T and 2.66T respectively. Assume that in round 1, node 'a' solves puzzles sequentially in order P1, P2, P3 and gets back with puzzle solutions in time 0.33T, 1.66T (0.33T + 1.33T) and 4.33T (0.33T + 1.33T + 3T) respectively. The computation speed of nodes 'a', 'b' and 'c' can be estimated to be 3X, 2.40X and 1.84X respectively (where X is the computation speed of node S) .

Suppose that in a subsequent round (say round 2), nodes 'a', 'b' and 'c' are issued puzzles of complexity level C3, C2 and C1 (call these puzzles P4, P5 and P6) respectively, and node 'a'

solves the puzzles sequentially in order P4, P6, P5 and responds in time 2.66T, 3T (2.66T + 0.33T) and 4.33T (2.66T + 0.33T + 1.33T) respectively. The computation speed of nodes 'a', 'b' and 'c' can be estimated to be 3X, 0.92X and 0.33X respectively. The computation speed of nodes 'b' and 'c' from round 1 and round 2 vary greatly and thus the network can suspect a Sybil attack.

Heuristic 2: Order only heuristic

An honest node will always end up responding to puzzles of same complexity level in same relative order in subsequent rounds of Sybil detection procedure. As the puzzles are *monolithic*, a malicious node does not have any idea about the hardness of a puzzle. If a compromised node assuming multiple identities (and thus receiving multiple puzzles) solves the puzzles sequentially, by randomly choosing amongst the puzzles it received, it might respond to puzzles of same complexity level in different relative orders in subsequent rounds, as explained in example below.

Example: Let S denote the node with slowest computation speed. Suppose that a compromised node 'a' with computation speed 3 times of the computation speed of the slowest node in the network assumes 3 identities 'a', 'b' and 'c'. As a result, it has to respond to puzzles issued to 'a', 'b' and 'c'. Assume that a verifier issues 3 puzzles (of type 2) P1, P2 and P3 of complexity level C1, C2 and C3 to nodes 'a', 'b' and 'c' respectively. Suppose that the puzzles with complexity level C1, C2 and C3 can be solved by node S in time T, 4T and 8T respectively. As the puzzles are monolithic, a compromised node cannot predict the hardness of a puzzle. Assume that a compromised node 'a' solves the puzzles sequentially and responds to puzzles in order P3, P2 and P1 in one round (which means node 'c' is faster than node 'b' and node 'b' is faster than node 'a'). Suppose that in a subsequent round, when a verifier issues puzzles of complexity level C1, C2 and C3 to nodes 'a', 'b' and 'c' (call these P4, P5 and P6) respectively, if node 'a' replies in a different order (say P4, P5, P6), then the network can suspect a Sybil attack. This will happen only when a compromised node randomly picks up puzzles to solve without considering to whom they are assigned. However, if node 'a' always solves the puzzles assigned to 'a' , 'b' and 'c' in that order for all the rounds, this Heuristic will not be able to suspect a Sybil attack.

The Table 5.1 below explains various approaches that could be adopted by a compromised node to respond to puzzles. These approaches are described in detail in later discussion. Suppose that a verifier has 'N' neighbors and a compromised node assumes identity of 'q' nodes. Let 'p' be the ratio of the computation speed of fastest to the computation speed of the slowest node in the network and 'T' be the time required by the slowest node in the network (denoted by S) to solve the easiest puzzle (of complexity level 0).

Following are the puzzle distributions used by a verifier as explained in table below to detect a Sybil attack.

Distribution 1: A verifier distributes N puzzles of type 2 which take time T, pT, (p^2) T.. (p^N) T to be solved by node S. The puzzle distribution is formed such that if a malicious node assuming multiple identities (and solving puzzles sequentially) solves a harder puzzle first, it will not be able to respond to an easier puzzle in time. This is explained in detail in cases 1 and 2 below.

Distribution 2: A verifier distributes N puzzles of type 1 with time distribution given by the equation, $t(P_{i+1}) = 2p * t(P_i)$ - $t(P_i)$, where $t(P_{i+1})$ is time required by node S to solve the puzzle of complexity level P_{i+1} and $t(P_i)$ is the time required by node S to solve the puzzle of complexity level P_i, P_0 = T, and 'p' is ratio of the computation speed of fastest to the computation speed of the slowest node in the network. The puzzle distribution is formed such that if a malicious node assuming multiple identities (and solving puzzles sequentially) solves a harder puzzle first, it will not be able to respond to an easier puzzle in time. This is explained in detail in cases 3 and 4 below.

Distribution 3: A verifier distributes N puzzles of type 2 which take time (linear distribution) T, 2T, 3T ... NT to be solved by node S.

Table 5.2: Table summarizing various approaches that could be followed by a malicious node assuming multiple identities to solve the puzzles

Approach by compromised node	**Puzzle Distribution**	**Defense Mechanism**	**Can a Sybil node escape detection?**	**How can a Sybil node escape detection?**
Case 1: A compromised node assuming 'q' identities solves puzzles sequentially and replies to a puzzle as soon as it gets the solution.	Distribution 1	Heuristic 1	Yes, with probability 1/q!	By solving the puzzles in strictly increasing order of complexity level

Continued on Next Page...

Table 5.2 – Continued

Approach by compromised node	**Puzzle Distribution**	**Defense Mechanism**	**Can a Sybil node escape detection?**	**How can a Sybil node escape detection?**
Case 2: A compromised node assuming 'q' identities solves puzzles sequentially and replies to all puzzles at once after solving all the puzzles.	Distribution 1	Heuristic 1	No	N/A
Case 3: A compromised node assuming 'q' identities solves puzzles sequentially and replies to a puzzle as soon as it gets the solution.	Distribution 2	Heuristic 1	Yes, with probability 1/q!	By solving the puzzles in strictly increasing order of complexity level
Case 4: A compromised node assuming 'q' identities solves puzzles sequentially and replies to all puzzles at once after solving all the puzzles.	Distribution 2	Heuristic 1	No	N/A
Case 5a: A compromised node assuming 'q' identities guesses an appropriate epoch interval 'T', and solves the puzzles in parallel during this epoch interval, and responds to puzzles as soon as it gets the solution. q >> p (for example q = 2p)	Distribution 3	Heuristic 1	No	N/A

Continued on Next Page...

Table 5.2 – Continued

Approach by compromised node	**Puzzle Distribution**	**Defense Mechanism**	**Can a Sybil node escape detection?**	**How can a Sybil node escape detection?**
Case 5b: A compromised node assuming 'q' identities guesses an appropriate epoch interval 'T', and solves the puzzles in parallel during this epoch interval, and responds to puzzles as soon as it gets the solution. q > p (for example q = p + 1)	Distribution 3	Heuristic 1	No	N/A
Case 5c.i: A compromised node assuming 'q' identities guesses an appropriate epoch interval 'T', and solves the puzzles in parallel during this epoch interval. q <= p. During an epoch, a compromised node responds to puzzles as it solves.	Distribution 3	Heuristic 1	No	N/A

Continued on Next Page...

Table 5.2 – Continued

Approach by compromised node	**Puzzle Distribution**	**Defense Mechanism**	**Can a Sybil node escape detection?**	**How can a Sybil node escape detection?**
Case 5c.ii: A compromised node assuming 'q' identities guesses an appropriate epoch interval 'T' and solves the puzzles in parallel during this epoch interval. q <= p. During an epoch, a compromised node responds to puzzles only during the end of an epoch	Distribution 1 or 3	Heuristic 1	Yes	A compromised node will be able to escape the detection procedure if it, 1. Correctly guess epoch to be time 'T'. 2. Correctly guesses the value of 'p'.
Case 6: A compromised node assuming 'q' identities gets the puzzles solved from an external attacker server.	Distribution 1 or Distribution 3	Heuristic 1	Yes	A compromised node will be able to escape the detection procedure if it, 1. Correctly guess epoch to be time 'T'. 2. Correctly guesses the value of 'p' (in this case ratio of speed of server to slowest node in the network).

The following discussion provides a detailed explanation of the cases summarized in the Table 5.2 above.

If a compromised node assumes identity of 'q' nodes, it will receive 'q' puzzles. A compromised node can solve these puzzles either sequentially or in parallel. Let S denote the node with slowest computation speed. Let 'p' be the ratio of the computation speed of the fastest node to the computation speed of the slowest node in the network and suppose that a verifier has

'N' neighbors. In each case, assume that an attacker manages to compromise the fastest node in the network.

Case 1: *A compromised node assuming multiple identities solves puzzles sequentially and replies to a puzzle as soon as it solves the puzzle. Verifier distributes puzzles of type 2.*

Puzzle distribution to thwart the attack: A verifier distributes 'N' puzzles of type 2 that require time T, pT, p^2 T ... p^NT to be solved by node S. ('T' is the time taken by node S to solve the easiest puzzle). As explained with an example below, the puzzles are generated such that if a compromised node assuming multiple identities solves the puzzles sequentially, and solves a harder puzzle first, it would not be able to respond to an easier puzzle in time.

If a compromised node assumes identity of 'q' nodes ($q <= p$ or $q > p$ or $q >> p$), and solves the puzzles sequentially, and replies to a puzzle as soon as it gets the solution to a puzzle, it would be able to escape the Sybil detection protocol with probability 1/q!. A compromised node will escape the detection procedure if and only if it solves the puzzles in strictly increasing order of complexity level.

The drawback of this puzzle distribution is that, if the value of 'N' is large (for example $N = 20$), the nodes that get harder puzzles will have to spend a lot of time in solving puzzles ($3486784401T$ is the time required to solve the hardest puzzle for $p = 3$). Thus nodes in the sensor network will spend a lot of its battery power computing solutions to puzzles, which is not desirable. So if 'N' is large, a verifier should randomly test 'M' nodes of its 'N' neighbors in each of the Sybil detection rounds and try to find Sybil nodes amongst them. The problem of how a verifier chooses 'M' most suspicious nodes out of 'N' nodes is out of the scope of our work.

Example: Let $p = 3$. Suppose that a compromised node 'a' with computation speed 3 times of the computation speed of the slowest node in the network assumes 3 identities 'a', 'b' and 'c'. A verifier issues puzzles of type 2 such that node 'a' receives 3 puzzles P1, P2 and P3 (actually issued to nodes 'a', 'b' and 'c' respectively) of complexity level C1, C2 and C3 which take time T, 3T and 9T to be solved by the node S. Node 'a' can solve the puzzles in 6 (=3!) possible orders ([P1, P2, P3], [P1, P3, P2], [P2, P1, P3], [P2, P3, P1], [P3, P1, P2], [P3, P1, P1]). However node 'a' will be able to respond to all the puzzles in time, and thus escape the Sybil detection procedure, if it solves the puzzles in increasing order of complexity level (P1, P2, P3). However, as the puzzles are monolithic, node 'a' does not know the easiest puzzle among P1, P2 and P3. As shown in the Table 5.3 below, if node 'a' chooses to solve puzzles in any order other than [P1, P2, P3], it will not be able to respond to all the puzzles in time. For example, if the node 'a' chooses to solve the puzzles in order [P2, P1, P3], it would respond to P1, P2 and P3 by 1.33T, T and 4.33T respectively. Thus it is late in responding to puzzle P1 and a Sybil attack would be suspected/detected (using Heuristic 1).

Table 5.3: Table illustrating Case 1

Order in which node 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack detected?
P1 P2 P3	T	0.33T	3T	1.33T	9T	4.33T	No
P1 P3 P2	T	0.33T	3T	4.33T	9T	3.33T	Yes
P2 P1 P3	T	1.33T	3T	T	9T	4.33T	Yes
P2 P3 P1	T	4.33T	3T	T	9T	4T	Yes
P3 P1 P2	T	3.33T	3T	4.33T	9T	3T	Yes
P3 P2 P1	T	4.33T	3T	4T	9T	3T	Yes

Case 2: *A compromised node assuming multiple identities solves puzzles sequentially and replies to all puzzles at once in increasing order of complexity, when it solves all the puzzles completely. Verifier distributes puzzles of type 2.*

Puzzle distribution to thwart the attack: A verifier distributes 'N' puzzles of type 2 that require time T, pT, (p^2) T... (p^N)T to be solved by node S.

If a compromised node assumes identity of 'q' nodes ($q <= p$ or $q > p$ or $q >> p$) and solves the puzzles sequentially, and replies to all puzzles at once when it solves all of them, it would not be able to reply to most of the puzzles in time (depending on 'q'). As shown in the Table 5.4 below, a malicious node would respond to all the puzzles at or after time 4.33T and hence it would be late in responding to P1 and P2. Thus a Sybil attack would be suspected/detected using Heuristic 1.

Table 5.4: Table illustrating Case 2

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack detected?
P1 P2 P3	T	4.33T	3T	4.33T	9T	4.33T	Yes
P1 P3 P2	T	4.33T	3T	4.33T	9T	4.33T	Yes
P2 P1 P3	T	4.33T	3T	4.33T	9T	4.33T	Yes
P2 P3 P1	T	4.33T	3T	4.33T	9T	4.33T	Yes
P3 P1 P2	T	4.33T	3T	4.33T	9T	4.33T	Yes
P3 P2 P1	T	4.33T	3T	4.33T	9T	4.33T	Yes

Case 3: *A compromised node assuming multiple identities solves puzzles sequentially and replies to a puzzle as soon as it solves the puzzle. Verifier distributes puzzles of type 1.*
Let S denote the node with slowest computation speed. As explained in Chapter 3, puzzles of type 1 require an average case analysis while analyzing the response time, since the generator does not know the exact time required to solve the puzzles (the generator only knows the best and the worst case time). For example, a puzzle P1 of complexity level 1 requiring time 'T' to be solved in worst case by node S, would require time T/2 to be solved in an average case. A node which is 'p' times computationally faster than the node S would solve puzzle P1 in time T/(2p), on an average case.
Puzzle distribution to thwart the attack: A verifier distributes puzzles of type 1 that have time complexity distribution given by the equation $t(P_{i+1}) = 2\times p\times t(P_i)$ - $t(P_i)$, where $t(P_{i+1})$ is the worst case time required by node S to solve the puzzle of complexity P_{i+1} and $t(P_i)$ is the worst case time required by node S to solve the puzzle of complexity P_i, P_0 = T, and 'p' is the ratio of the computation speed of the fastest to the computation speed of the slowest node in the network. As explained with an example below, the puzzles are generated such that if a compromised node assuming multiple identities solves the puzzles sequentially, and solves a harder puzzle first, then it would not be able to respond to an easier puzzle in time, on an average case.
Explanation for puzzle distribution: Let $t(P1)$ denote the worst case time required by node S to solve the puzzle P1 of complexity level 1, and $t(P2)$ denote the worst case time required by node S to solve the puzzle P2 of complexity level 2. *The puzzles should be generated such*

that, if an attacker solves a harder puzzle before an easier puzzle (say P2 before P1), then it should not be able to solve the easier puzzle (P1) in time, considering the average time to solve both P1 and P2 (statement 1). The time required to solve P2 varies between $t(P1)$ (best case) and $t(P2)$ (worst case), so on an average the time required to solve P2 is $(t(P1) + t(P2))/2$. A node with a computation speed 'p' times that of node S can solve the puzzle P2 in time $(t(P1)+t(P2))/2p$, on an average. So for statement 1 to be true, $(t(P1)+t(P2))/2p >= t(P1)$, so $t(P2) = 2p * t(P1) - t(P1)$. Generalizing, $t(P_{i+1}) = 2p * t(P_i) - t(P_i)$.

If a compromised node assumes identity of 'q' nodes ($q <= p$ or $q > p$ or q>>p) and solves the puzzles it receives sequentially, and replies to a puzzle as soon as it gets the solution to the puzzle, it would be able to escape the Sybil detection protocol with probability 1/q!. A compromised node assuming multiple identities will escape the Sybil detection procedure if and only if it solves the puzzles in strictly increasing order of complexity level.

Example: Let p = 3. Suppose that a compromised node 'a' with computation speed 3 times of the computation speed of the slowest node in the network assumes 3 identities 'a', 'b' and 'c'. A verifier issues puzzles of type 1 such that node 'a' receives 3 puzzles P1, P2 and P3 (issued to nodes 'a', 'b' and 'c' respectively) of complexity level C1, C2 and C3, which take time T, 5T and 25T to be solved by the node S, in worst case. The puzzle complexity distribution is such that, if a node solves a harder puzzle (puzzle with higher complexity level) first, it would not be able to respond to an easier puzzle in time. Node 'a' can solve the puzzles in 6 possible orders. However, as shown in the Table 5.5 below, it can only escape from being detected if it solves puzzles in increasing order of complexity level. Puzzles P1, P2 and P3 can be solved in time T/2, 3T [(T + 5T)/2] and 15T [(5T + 25T)/2] in average case by the node S. Node 'a' would solve these puzzles in time T/6, T and 5T respectively (since it is 3 times faster than node S). For example, if node 'a' solves the puzzles in order [P2, P1, P3], it would not be able to respond to P1 in time, on an average case. As shown in the Table 5.5 below, if node 'a' solves the puzzles in any order except increasing order of complexity, a Sybil attack would be detected (using Heuristic 1) since node 'a' would not be able to respond to all the puzzles in time (on an average case).

Table 5.5: Table illustrating Case 3

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack detected?
P1 P2 P3	T	0.16T	5T	1.16T	25T	6.16T	No
P1 P3 P2	T	0.16T	5T	6.16T	25T	5.16T	Yes
P2 P1 P3	T	1.16T	5T	T	25T	6.16T	Yes
P2 P3 P1	T	6.16T	5T	T	25T	6T	Yes
P3 P1 P2	T	5.16T	5T	6.16T	25T	5T	Yes
P3 P2 P1	T	6.16T	5T	6T	25T	5T	Yes

Case 4: *A compromised node assuming multiple identities solves puzzles sequentially and replies to all puzzles at once in increasing order of complexity, when it solves all of them completely. Verifier distributes puzzles of type 1.*

Puzzle distribution to thwart the attack: A verifier distributes puzzles of type 1 that have time complexity distribution given by the equation $t(P_{i+1})$= 2×p× $t(P_i)$ - $t(P_i)$, where $t(P_{i+1})$ is time required by node S to solve the puzzle of complexity level P_{i+1} in worst case and $t(P_i)$ is the time required by node S to solve the puzzle of complexity level P_i in worst case, P_0 = T, and 'p' is the ratio of the computation speed of the fastest to the computation speed of the slowest node in the network.

If a compromised node assumes identity of 'q' nodes ($q <= p$ or $q > p$ or q>>p) and solves the puzzles it receives sequentially, and replies to all puzzles at once when it solves all of them, it would not be able to reply to most of the puzzles in time, irrespective of the order in which it chooses to solve the puzzles. As shown in the Table 5.6 below, the compromised node will respond to all puzzles at once, at or after time 6.16T. Thus a compromised node is late in responding to P1 and P2 and a Sybil attack would be suspected/detected (using Heuristic 1).

Table 5.6: Table illustrating Case 4

Order in which 'a' solves the puzzles	Node 'a' is expected to respond latest by	Node 'a' responds by	Node 'b' is expected to respond latest by	Node 'b' responds by	Node 'c' is expected to respond latest by	Node 'c' responds by	Sybil attack detected?
P1 P2 P3	T	6.16T	5T	6.16T	25T	6.16T	Yes
P1 P3 P2	T	6.16T	5T	6.16T	25T	6.16T	Yes
P2 P1 P3	T	6.16T	5T	6.16T	25T	6.16T	Yes
P2 P3 P1	T	6.16T	5T	6.16T	25T	6.16T	Yes
P3 P1 P2	T	6.16T	5T	6.16T	25T	6.16T	Yes
P3 P2 P1	T	6.16T	5T	6.16T	25T	6.16T	Yes

Case 5: *A compromised node assuming multiple identities guesses an appropriate epoch interval and solves the puzzles in parallel during this epoch interval. The epoch interval should be a divisor of the time 'T' where 'T' is the time required by the node S to solve the easiest puzzle. T = m*E, (m >=1).*

Puzzle distribution to thwart the attack: A verifier distributes puzzles of type 2 (of linear complexity) which take time T, 2T, 3T... NT to be solved by the node S.

Suppose that a compromised node assumes identity of 'q' nodes. An attacker node creates a thread for each of the puzzle ('q' threads in total). A compromised node guesses an appropriate epoch interval 'E' (which should be at most 'T', where 'T' is the time required by node S to solve the easiest puzzle) and solves each puzzle for E/q time (during the epoch interval) and then switches to another puzzle. Let 'p' be the ratio of the computation speed of the fastest node in the network to the computation speed of node S.

Case a]: *q >> p (for example q = 2p, let p=3, q=6 and E = T).* Suppose that a compromised node 'a' assumes identity of 6 nodes 'a', 'b', 'c', 'd', 'e' and 'f'. The compromised node would create a thread for each puzzle and solve each puzzle for time, T/q = T/2p. Assume that the compromised node gets puzzles P1, P2 ... P6 which take time T, 2T ... 6T.

In epoch 1 (till time T), the compromised node ('a') will allocate T/6 units of time to each puzzle and since its computation speed is 3 times that of node S, it will complete T/2 portion of work for each of the unsolved puzzles. In epoch 2 (till time 2T), node 'a' will allocate T/6 units of time to each puzzle and will complete T (T/2 from 1st epoch and T/2 from 2nd epoch) portion of work for each of the unsolved puzzles. It completes solving puzzle P1 and replies

with solution to P1. But it is late in replying to puzzle P1 (since it replies by time 2T). Now node 'a' has 5 more puzzles to solve. In epoch 3 (till time 3T), node 'a' will allocate T/5 units of time to each puzzle and since its computation speed is 3 times that of node S, it will complete 1.6T (T/2 from 1st epoch and T/2 from 2nd epoch and 0.6T from 3rd epoch) portion of work for each of the unsolved puzzle. If we continue the analysis, we would find that the node 'a' will be able to respond to P2 in epoch 4, P3 in epoch 6, P4 in epoch 7 and P5, P6 in epoch 8. Thus node 'a' will not be able to respond to most of the puzzles in time and a Sybil attack will be suspected/detected using Heuristic 1. It can be argued that out of the 'N' puzzles issued by a verifier, a compromised node might get puzzles that do not follow a linear distribution. For example, if $N = 20$, a compromised node can get puzzles that take time T, 3T, 9T, 12T, 15T, 20T. A compromised node will still not be able to respond to most of the puzzles in time.

Case b]: *$q > p$ (for example $q = p+1$, let $p=3$, $q=4$ and $E = T$).*

Example: Suppose that a compromised node 'a' assumes identity of 4 nodes 'a', 'b', 'c' and 'd'. Node 'a' would create a thread for each puzzle and solve each puzzle for time, T/q = T/(p+1) within the epoch interval. Suppose that node 'a' gets puzzles P1, P2 ... P4 of type 2 which take time T, 2T ... 4T to be solved by node S.

In epoch 1 (till time T), the node 'a' will allocate T/4 time to each puzzle and since it is 3 times computationally faster than node S, it will complete 3T/4 portion of work for each puzzle. In epoch 2 (till time 2T), node 'a' will allocate T/4 time to each puzzle and since it is 3 times computationally faster than node S, it will complete 1.5T portion of work for each unsolved puzzle (0.75T each from epoch 1 and 2 respectively). Mean while, it will complete solving and respond to puzzle P1. But it is late in responding to puzzle P1. Now node 'a' has 3 more puzzles to solve. In epoch 3 (till time 3T), node 'a' will allocate T/3 time to each puzzle and since it is 3 times computationally faster than node S, it will complete 2.5T portion of work for each unsolved puzzle (0.75T from epoch 1 and 2, and T from epoch 3). Mean while it will solve P2 and respond to P2. But it is late in responding to P2. Node 'a' has 2 more puzzles to solve and it would be able to respond to P3 and P4 in epoch 4. Thus node 'a' is not able to respond to P1, P2 and P3 in time and hence an attack would be suspected/detected (using Heuristic 1).

Case c.i]: *$q <= p$. During an epoch, a compromised node responds to puzzles as soon as it solves the puzzle.*

In this case by measuring the exact time at which the puzzle solutions are received, the speed of the node can be estimated and inconsistencies in subsequent speed estimations can be used to detect a Sybil attack.

Example: Suppose a compromised node 'a' assuming 3 identities 'a', 'b' and 'c' gets puzzles P1, P2, P3 (of type 2) respectively, which take time T, 2T, 3T to be solved by the node S.

In epoch 1 (till time T), node 'a' will allocate T/3 time to each puzzle and since it is 3 times

computationally faster than node S, it will complete T portion of work for each puzzle. If the node 'a' solves puzzles in order [P1, P2, P3] or [P1, P3, P2] during an epoch, it can respond to P1 in T/3 time. So speed of identity 'a' can be estimated to be 3X (where X is the computation speed of node S). Similarly, if the node 'a' solves puzzles during an epoch in order [P2, P1, P3] or [P3, P1, P2], it can respond to P1 in 2T/3 time. So the speed of identity 'a' can be estimated to be 1.5X. Similarly, if node 'a' solves puzzles during an epoch in order [P2, P3, P1] or [P3, P2, P1], it can respond to P1 in T time. So the speed of identity 'a' can be estimated to be X. The compromised node will continue to solve the puzzles P2 and P3 in subsequent epochs.
Suppose that in a subsequent round, 'a', 'b' and 'c' get puzzles P4, P5, P6 respectively, which take time 2T, T, 3T to be solved by node S. In epoch 1 (till time T), the compromised node will allocate T/3 time to each puzzle and since it is 3 times computationally faster than the node S, it will complete T portion of work required for each puzzle. Thus the compromised node can complete solving puzzle P4 at time 1.33T, 1.66T or 2T depending on the order in which it solves puzzles during an epoch. Thus the speed of identity 'a' will be estimated to be 1.5X, 1.2X or X.
For a large number of rounds depending on the order in which puzzles are solved, the estimated speed of identities which the compromised node masquerades might vary, which can be an indication of Sybil attack (using extension to Heuristic 1).
Case c.ii): *During an epoch, a compromised node responds with puzzle solutions only at the end of an epoch.*
Assuming that an attacker manages to compromise one of the faster nodes in the network, a compromised node would be able to assume identity of 'q' nodes without being detected, if it correctly guesses its computation speed (the ratio of the computation speed of the compromised node to the computation speed of node S) relative to node S, and also epoch interval E (E = T/m, m >= 1). However, the values 'p' and 'T' are not known to any node in the network. Also, as each node's identity is associated with a unique identity key, an attacker will not be able to fabricate new identities. So even if the attacker manages to compromise the fastest node in the network and correctly guess the node's computation speed relative to node S, and correctly guesses the epoch interval E, it would be able to successfully assume identity of 'q' nodes without being detected only if it manages to compromise 'q' legitimate nodes to read off the identity key from those nodes.
Case 6: *A compromised node gets the puzzles solved from an external computationally powerful attacker server.*
It can be argued that an attacker can have infinite (extremely large) computation power and hence be able to masquerade a large number of nodes. An attacker can compromise a node and may connect it via a low latency medium (like ethernet) to an external powerful server (in worst case). Thus technically, a compromised node will have infinite computation power. However,

to masquerade a large number of nodes ('n'), an attacker node needs to have the identity key of 'n' legitimate nodes (compromising a large number of nodes is difficult). Also, a verifier having a large number of neighbors as compared to an expected normal may be an indication of a Sybil attack in some networks (A verifier may have a large number of neighbors if, one of the neighbor is compromised and assuming large number of identities). Thus an attacker node will be able to masquerate 'n' nodes ('n' is large) only if the attacker manages to compromise the identity key of 'n' legitimate nodes.

5.4 Synchronization to Augment Robustness

We propose the synchronization protocol to prevent the below two conditions from occurring,

1. **Puzzle replay attack:** A target node can replay the puzzles and get the puzzles solved from their neighbors. So even though a node is a Sybil, it will always be able to respond to all the puzzles in time. We call this approach taken by a Sybil or any malicious node as 'Puzzle replay attack' approach.

2. **Timing synchronization for puzzles:** A target can receive puzzles from multiple verifiers at a time. In that case, a target will not be able to respond to all the puzzles in time. Even though the target node is not a Sybil, the protocol might falsely detect the target node to be a Sybil.

The protocol data units (PDU) used in the synchronization protocol are explained below:
Protocol PDU definitions:
Puzzle send PDU: Broadcasted by a verifier to targets when a verifier (V) intends to issue puzzles to targets. The PDU is encrypted using a network wide shared key. This is send by the verifier to all of its neighbors that are discovered during the neighbor discovery protocol. Denoted later as PDU1.
Puzzle send Ok PDU: Sent from a target to a verifier in response to PDU1. All the neighbors of verifier V should respond with this packet. Denoted later as PDU2.
Puzzle question PDU: This packet conveys the puzzle question from a verifier to a specific target. Denoted later as PDU3.
Puzzle solution PDU: This packet is sent by the target to the verifier giving the puzzle solution. Denoted later as PDU4.

The messages in synchronization protocol are exchanged over a network wide shared key. Thus nodes can overhear each other. The synchronization protocol is explained using example in figure 5.1 below. In the figure 5.1 below, consider: Nodes (1, 2, 3, 4, 5) = Targets, V= Verifier, N = Neighbors of targets which are itself neither targets nor a verifier. The verifier

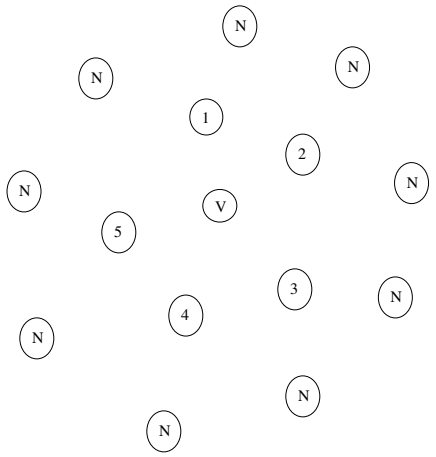


Figure 5.1: A figure to illustrate Synchronization protocol

and other neighbors of target (nodes marked as N) will monitor if any of the target does not follow the synchronization protocol and report this information to the base station.

1. A verifier (V) broadcasts PDU1 to all of its neighbors (target nodes 1 through 5).
2. All of the neighbors (target nodes 1 through 5) should respond with their readiness to receive the puzzle (PDU 2).
3. The neighbors of target nodes 1, 2, 3, 4 and 5 (nodes marked as N) will realize that their neighbors (target nodes 1 through 5) are solving puzzles. So nodes marked as N will not solve any puzzles from nodes 1 through 5 and will not issue them puzzles.
4. Verifier issues a PDU3 to each of the target (nodes 1 through 5).
5. Whenever the targets (nodes 1 through 5) respond with puzzle solutions (PDU 4), the neighbors of the targets (nodes marked as N) can solve puzzles from those targets (nodes 1 through 5) or issue them puzzles.

Chapter 6

Results, Conclusion and Future work

6.1 Results: Homogeneous sensor network

In case of a homogeneous sensor network, all the nodes have same computing capabilities. We implemented the various approaches that could be followed by a compromised node (assuming multiple identities) on a custom built simulator and also on droid phones.
A malicious node assuming multiple identities will receive multiple puzzles. Following are the approaches that can be followed by a malicious node assuming multiple identities to solve the puzzles.

1. **Approach 1:** A compromised node assuming multiple identities solves puzzles sequentially. Replies with all the puzzle solutions at once, after completely solving all the puzzles.

2. **Approach 2:** A compromised node assuming multiple identities solves puzzles sequentially. Replies with a puzzle solution as soon as it gets the solution to the puzzle.

3. **Approach 3:** A compromised node assuming multiple identities solves puzzles in parallel. Replies with all the puzzle solutions at once, after completely solving all the puzzles.

4. **Approach 4:** A compromised node assuming multiple identities solves puzzles in parallel. Replies with a puzzle solution as soon as it gets the solution to the puzzle.

6.1.1 Results with Custom simulator

Simulation 1: In this simulation, we assume that a malicious node assumes identity of 2 legitimate nodes. A verifier sensor node distributes puzzles of type 2 of same complexity level and each puzzle requires time T (= 8 seconds) to be solved by each node in the network (since all the nodes have same computation capability). As expected, a malicious node is not able to

respond with all the puzzle solutions in time. Following are the results based on 100 runs of Sybil detection procedure for each of the approaches explained above.

Table 6.1: Table summarizing results for Simulation 1

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Percentage Detection
Approach 1	100	100	100%
Approach 2	100	100	100%
Approach 3	100	100	100%
Approach 4	100	100	100%

Simulation 2: In this simulation, we assume that a malicious node assumes identity of 3 legitimate nodes. A verifier sensor node distributes puzzles of type 2 of same complexity level and each puzzle requires time T (= 8 seconds) to be solved by each node in the network. As expected, a malicious node assuming multiple identities is not able to respond with all the puzzle solutions in time. Following are the results based on 100 runs of Sybil detection procedure for each of the approaches explained above.

Table 6.2: Table summarizing results for Simulation 2

Approach by a malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Percentage Detection
Approach 1	100	100	100%
Approach 2	100	100	100%
Approach 3	100	100	100%
Approach 4	100	100	100%

6.1.2 Case study with Android phones

We also implemented the Sybil detection procedure on Droid phones. The Droid phones were connected through WiFi.

Implementation 1: Suppose that a malicious droid assumes identity of 2 legitimate nodes. A verifier droid phone distributes puzzles of type 2 of same complexity level and each puzzle requires time T (= 8 seconds) to be solved by each droid phone (since all the nodes have same computation capability). As expected, a malicious droid node is not able to respond with all

the puzzle solutions in time. Following are the results based on 100 runs of Sybil detection procedure for each of the approaches explained above.

Table 6.3: Table summarizing results for Implementation 1

Approach by a malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Percentage Detection
Approach 1	100	100	100%
Approach 2	100	100	100%
Approach 3	100	100	100%
Approach 4	100	100	100%

Implementation 2: Suppose that a malicious droid assumes identity of 3 legitimate nodes. A verifier droid distributes puzzles of type 2 of same complexity level and each puzzle requires time T (= 8 seconds) to be solved by each droid phone (since all the nodes have same computation capability). As expected, a malicious droid node is not able to respond with all the puzzle solutions in time. Following are the results based on 100 runs of Sybil detection procedure for each of the approaches explained above.

Table 6.4: Table summarizing results for Implementation 2

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Percentage Detection
Approach 1	100	100	100%
Approach 2	100	100	100%
Approach 3	100	100	100%
Approach 4	100	100	100%

6.1.3 False Positives

The tables below summarize the false positives in case of homogeneous sensor network. The guard time indicates an extra buffer time to account for slightly different processing speeds.

Table 6.5: False positives: Puzzle type 1: complexity level 1

Puzzle type	Number of runs	Puzzle complexity level	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	1	0.5	0	0%
1	50	1	0.25	0	0%

Table 6.6: False positives: Puzzle type 1: complexity level 2

Puzzle type	Number of runs	Puzzle complexity level	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	2	1	0	0%
1	50	2	0.5	0	0%
1	50	2	0.25	0	0%

Table 6.7: False positives: Puzzle type 1: complexity level 3

Puzzle type	Number of runs	Puzzle complexity level	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	3	2	0	0%
1	50	3	1.5	0	0%
1	50	3	1	0	0%

Table 6.8: False positives: Puzzle type 2: complexity level 1

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	8	2	0	0%
2	50	8	1	0	0%
2	50	8	0.5	2	4%
2	50	8	0.25	5	10%

Table 6.9: False positives: Puzzle type 2: complexity level 2

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	4	1	0	0%
2	50	4	0.5	2	4%
2	50	4	0.25	3	6%

6.2 Results: Heterogeneous sensor network

In case of a heterogeneous sensor network, nodes may have different computation capability. We implemented the various approaches that could be followed by a compromised node (assuming multiple identities) on a custom built simulator and also on droid phones. Let S denote the node with slowest computation speed. Let 'p' denote the ratio of the computation speed of the fastest node to the computation speed of the slowest node in the network and 'q' denote the number of identities assumed by a malicious node.
Assuming that a verifier has N neighbors, following are the puzzle distributions used to thwart the Sybil attack.

1. **Distribution 1:** A verifier distributes N puzzles of type 2 that take time T, pT, (p^2) T.. (p^N) T to be solved by node S.

2. **Distribution 2:** A verifier distributes N puzzles of type 1 with time distribution given by the equation, $t(P_{i+1}) = 2p * t(P_i) - t(P_i)$, where $t(P_{i+1})$ is the worst case time required by node S to solve a puzzle of complexity P_{i+1} and $t(P_i)$ is the worst case time required

by node S to solve a puzzle of complexity P_i, P_0 = T and 'p' is ratio of the computation speed of fastest to the computation speed of the slowest node in the network.

3. **Distribution 3:** A verifier distributes N puzzles of type 2 that take time T, 2T, 3T ... NT to be solved by node S.

A malicious node assuming multiple identities will receive multiple puzzles. Following are the approaches that can be followed by a malicious node assuming multiple identities to solve the puzzles.

1. **Case 1:** A compromised node assuming multiple identities solves puzzles sequentially and replies to a puzzle as soon as it solves the puzzle. Verifier distributes puzzles of type 2, following Distribution 1.

2. **Case 2:** A compromised node assuming multiple identities solves puzzles sequentially and replies to all puzzles at once in increasing order of complexity, when it solves all the puzzles completely. Verifier distributes puzzles of type 2, following Distribution 1.

3. **Case 3:** A compromised node assuming multiple identities solves puzzles sequentially and replies to a puzzle as soon as it solves the puzzle. Verifier distributes puzzles of type 1, following Distribution 2.

4. **Case 4:** A compromised node assuming multiple identities solves puzzles sequentially and replies to all puzzles at once in increasing order of complexity, when it solves all of them completely. Verifier distributes puzzles of type 1, following Distribution 2.

5. **Case 5a:** A compromised node assuming multiple identities guesses an appropriate epoch interval and solves the puzzles in parallel during this epoch interval. The epoch interval should be a divisor of the time (T) required by the slowest node to solve the easiest puzzle. T = m*E, (m >=1). $\mathbf{q} >> \mathbf{p}$. Verifier distributes puzzles of type 2, following Distribution 3.

6. **Case 5b:** A compromised node assuming multiple identities guesses an appropriate epoch interval and solves the puzzles in parallel during this epoch interval. The epoch interval should be a divisor of the time (T) required by the slowest node to solve the easiest puzzle. T = m*E, (m >=1). $\mathbf{q} > \mathbf{p}$. Verifier distributes puzzles of type 2, following Distribution 3.

7. **Case 5c:** A compromised node assuming multiple identities guesses an appropriate epoch interval and solves the puzzles in parallel during this epoch interval. The epoch interval should be a divisor of the time (T) required by the slowest node to solve the easiest puzzle.

T = m*E, (m >=1). **q <= p**. Verifier distributes puzzles of type 2, following Distribution 3.

6.2.1 Results with Custom simulator

Simulation 3: Let S denote the node with slowest computation speed. In this simulation, we assume that a malicious node is 2 times computationally faster ($p = 2$) than node S. Let 'q' denote the number of identities assumed by a malicious node.
For cases 1 and 2, suppose that a malicious node assuming 2 identities gets 2 puzzles of Type 2, that take time 2 seconds and 4 seconds to be solved by node S. For cases 3 and 4, suppose that a malicious node assuming 2 identities gets 2 puzzles of Type 1, that take time 2 seconds and 6 seconds to be solved by node S, in worst case. For cases 5.a, 5.b and 5.c, suppose that the malicious node gets puzzles following linear distribution (3, 4, 5 .... seconds) depending on the number of identities it assumes. Following are the results for each of the approaches explained above.

Table 6.10: Table summarizing results for Simulation 3

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Detection Percentage	Expected detection percentage	Solution time of puzzles issued to malicious node
Case 1 ($q = 2$)	100	52	52%	50%	(2, 4) sec
Case 2 ($q = 2$)	100	100	100%	100%	(2, 4) sec
Case 3 ($q = 2$)	100	53	53%	50%	(2, 6) sec
Case 4 ($q = 2$)	100	100	100%	100%	(2, 6) sec
Case 5.a ($q = 4$)	100	100	100%	100%	(2, 3, 4, 5) sec
Case 5.b ($q = 3$)	100	100	100%	100%	(2, 3, 4) sec
Case 5.c ($q = 2$)	100	0	0%	0%	(2, 3) sec

Simulation 4: Let S denote the node with slowest computation speed. In this simulation, we assume that a malicious node is 3 times computationally faster ($p = 3$) than node S. Let 'q' denote the number of identities assumed by a malicious node.
For cases 1 and 2, assume that a malicious node assuming 3 identities gets 3 puzzles of Type 2, that take time 3 seconds, 9 seconds and 27 seconds to be solved by node S. For cases 3 and 4, assume that a malicious node assuming 3 identities gets 3 puzzles of Type 1, that take time 2 seconds, 10 seconds and 50 seconds to be solved by node S. For case 5.a, 5.b and 5.c, suppose that the malicious node gets puzzles following linear distribution (3, 4, 5 .... seconds) depending

on the number of identities it assumes. Following are the results for each of the approaches explained above.

Table 6.11: Table summarizing results for Simulation 4

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Detection Percentage	Expected detection percentage	Solution time of puzzles issued to malicious node
Case 1 ($q = 3$)	100	86	86%	84%	(3, 9, 27) sec
Case 2 ($q = 3$)	100	100	100%	100%	(3, 9, 27) sec
Case 3 ($q = 3$)	100	85	85%	84%	(2, 10, 50) sec
Case 4 ($q = 3$)	100	100	100%	100%	(2, 10, 50) sec
Case 5.a ($q = 6$)	100	100	100%	100%	(2...7) sec
Case 5.b ($q = 4$)	100	100	100%	100%	(2, 3, 4, 5) sec
Case 5.c ($q = 3$)	100	0	0%	0%	(2, 3, 4) sec

6.2.2 Case study with Android phones

We also implemented the Sybil detection algorithm on droid phones. The case of heterogeneous nodes was simulated by dividing the puzzle solution time with 'p' (where 'p' is the ratio of the computation speed of the fastest to the computation speed of the slowest node in the network), assuming that the computationally fastest node is compromised and masquerading multiple identities. The results were similar to our simulation results and as expected.

Implementation 3: Let S denote the node with slowest computation speed. We assume that a malicious droid node is 2 times computationally faster ($p = 2$) than node S. Let 'q' denote the number of identities assumed by a malicious node. For cases 1 and 2, suppose that a malicious node assuming 2 identities gets 2 puzzles of Type 2, that take time 2 seconds and 4 seconds to be solved by node S. For cases 3 and 4, suppose that a malicious node assuming 2 identities gets 2 puzzles of Type 1, that take time 2 seconds and 6 seconds to be solved by node S, in worst case. For case 5.a, 5.b and 5.c, suppose that a malicious node gets puzzles following linear distribution (3, 4, 5 .... seconds) depending on the number of identities it assumes. Following are the results for each of the approaches explained above.

Table 6.12: Table summarizing results for Implementation 3

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Detection Percentage	Expected detection percentage	Solution time of puzzles issued to malicious node
Case 1 ($q = 2$)	100	51	51%	50%	(2, 4) sec
Case 2 ($q = 2$)	100	100	100%	100%	(2, 4) sec
Case 3 ($q = 2$)	100	49	49%	50%	(2, 6) sec
Case 4 ($q = 2$)	100	100	100%	100%	(2, 6) sec
Case 5.a ($q = 4$)	100	100	100%	100%	(2, 3, 4, 5) sec
Case 5.b ($q = 3$)	100	100	100%	100%	(2, 3, 4) sec
Case 5.c ($q = 2$)	100	0	0%	0%	(2, 3) sec

Implementation 4: Let S denote the node with slowest computation speed. We assume that a malicious droid node is 3 times faster ($p = 3$) than node S. Let 'q' denote the number of identities assumed by a malicious node. For cases 1 and 2, suppose that a malicious node assuming 3 identities gets 3 puzzles of Type 2, that take time 3 seconds, 9 seconds and 27 seconds to be solved by node S. For cases 3 and 4, suppose that a malicious node assuming 3 identities gets 3 puzzles of Type 1, that take time 2 seconds, 10 seconds and 50 seconds to be solved by node S, in worst case. For cases 5.a, 5.b and 5.c, suppose that a malicious node gets puzzles following linear distribution (3, 4, 5 .... seconds) depending on the number of identities it assumes. Following are the results for each of the approaches explained above.

Table 6.13: Table summarizing results for Implementation 4

Approach by malicious node	Number of runs of Sybil detection procedure	Number of times attack detected	Detection Percentage	Expected detection percentage	Solution time of puzzles issued to malicious node
Case 1 ($q = 3$)	100	86	86%	84%	(3, 9, 27) sec
Case 2 ($q = 3$)	100	100	100%	100%	(3, 9, 27) sec
Case 3 ($q = 3$)	100	83	83%	84%	(2, 10, 50) sec
Case 4 ($q = 3$)	100	100	100%	100%	(2, 10, 50) sec
Case 5.a ($q = 6$)	100	100	100%	100%	(2...7) sec
Case 5.b ($q = 4$)	100	100	100%	100%	(2, 3, 4, 5) sec
Case 5.c ($q = 3$)	100	0	0%	0%	(2, 3, 4) sec

6.2.3 False Positives

The tables below summarize false positives in case of a heterogeneous sensor network. Let S denote the node with slowest computation speed (X). The guard time is an extra buffer time to account for slightly different processing times.

Table 6.14: False positives: Puzzle type 1: complexity level 1, Node speed 3X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	3	2	0	0%
1	50	3	1	0	0%
1	50	3	0.5	0	0%
1	50	3	0.25	0	0%

Table 6.15: False positives: Puzzle type 1: complexity level 2, Node speed 3X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	6	2	0	0%
1	50	6	1	0	0%
1	50	6	0.5	0	0%
1	50	6	0.25	0	0%

Table 6.16: False positives: Puzzle type 1: complexity level 1, Node speed 2X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	3	2	0	0%
1	50	3	1	0	0%
1	50	3	0.5	0	0%
1	50	3	0.25	0	0%

Table 6.17: False positives: Puzzle type 1: complexity level 2, Node speed 2X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
1	50	6	2	0	0%
1	50	6	1	0	0%
1	50	6	0.5	0	0%
1	50	6	0.25	0	0%

Table 6.18: False positives: Puzzle type 2: complexity level 1, Node speed 3X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	3	2	0	0%
2	50	3	1	0	0%
2	50	3	0.5	0	0%
2	50	3	0.25	0	0%

Table 6.19: False positives: Puzzle type 2: complexity level 2, Node speed 3X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	6	2	0	0%
2	50	6	1	0	0%
2	50	6	0.5	0	0%
2	50	6	0.25	0	0%

Table 6.20: False positives: Puzzle type 2: complexity level 1, Node speed 2X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	3	2	0	0%
2	50	3	1	0	0%
2	50	3	0.5	0	0%
2	50	3	0.25	0	0%

Table 6.21: False positives: Puzzle type 2: complexity level 2, Node speed 2X

Puzzle type	Number of runs	Generation time (seconds)	Guard time (seconds)	Number of times falsely detected	False positive percentage
2	50	6	2	0	0%
2	50	6	1	0	0%
2	50	6	0.5	0	0%
2	50	6	0.25	0	0%

6.2.4 Results: Extension to Heuristic 1

Let S denote the node with slowest computation speed. Suppose that a compromised node with computation speed 3 times the computation speed of node S, assumes 3 identities (node 0, node 1 and node 2) and solves the puzzles sequentially, and replies to a puzzle as soon as it solves the puzzle. Extension to Heuristic 1 attempts to estimate the speed of a node based on the time taken by the node to solve the puzzle. Suppose that puzzles P1, P2 and P3 (of type 2) issued to 3 identities take time 3, 9 and 27 seconds to be solved by node S. As the compromised node does not know the complexity level of puzzles, it might choose to solve the puzzles randomly. Table below summarizes the estimated computation speed of the nodes based on the timing at which the puzzle solutions are received, when a compromised node solves the puzzles sequentially by randomly picking amongst the puzzles it received. As shown below, if a node is malicious, its estimated speed will vary drastically for different rounds. The estimated computation speed is calculated relative to node S (Estimated speed of a node A = Puzzle solution time by node S/ Puzzle solution time by node A).

Table 6.22: Extension to Heuristic 1-malicious node

Order in which a compromised node solves the puzzles	Puzzles in column 1 assigned to nodes (respectively)	Estimated speed-node 0	Estimated speed-node 1	Estimated speed-node 2
P3 P2 P1	0 1 2	3.15	0.67	0.20
P3 P1 P2	1 2 0	0.68	2.97	0.29
P1 P3 P2	1 0 2	2.72	3.01	0.69
P1 P2 P3	1 2 0	2.11	2.99	2.30
P3 P1 P2	1 2 0	0.66	2.99	0.30
P1 P2 P3	0 1 2	2.96	2.54	1.68
P2 P1 P3	0 1 2	2.97	0.81	1.52
P3 P1 P2	2 1 0	0.99	0.45	3.09
P3 P1 P2	1 2 0	0.45	1.75	0.18
P1 P3 P2	1 2 0	0.68	2.98	2.67
P2 P1 P3	1 0 2	0.75	3.07	1.79
P1 P3 P2	0 1 2	3.02	2.83	0.72
P3 P2 P1	2 1 0	0.22	0.75	2.96
P1 P3 P2	2 0 1	2.69	0.68	3.01
P3 P2 P1	0 2 1	2.99	0.25	0.75
P3 P2 P1	2 0 1	0.75	0.23	3.00

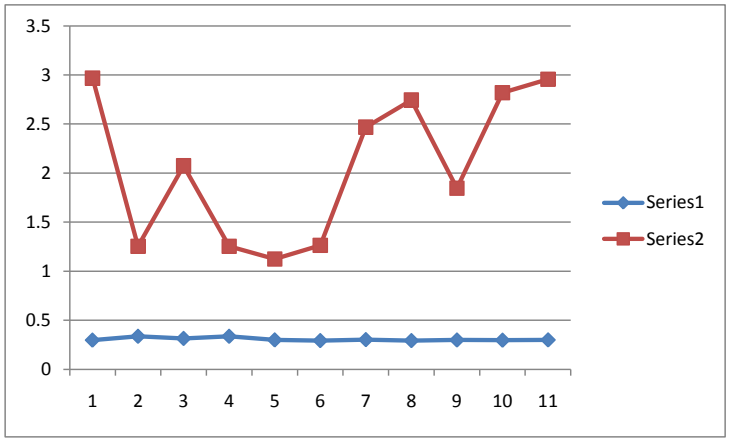
The table below shows that the estimated computation speed of honest node (A) relative to node S will always be consistent for subsequent rounds.

Table 6.23: Extension to Heuristic 1-Honest node

Puzzle assigned to node A	Actual Speed relative to node S	Estimated speed-node A
P1	3	2.97
P2	3	2.98
P1	3	3.00
P2	3	3.00
P1	3	3.08
P1	3	2.85
P3	3	3.02
P3	3	3.02
P2	3	2.99
P1	3	2.97
P1	3	3.01
P2	3	2.96
P3	3	2.96

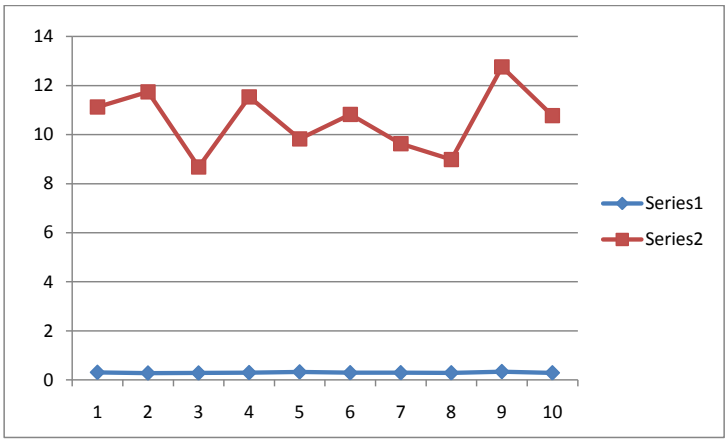
6.3 Results: Puzzles

The following plots compare the generation time and the solution time for puzzles of different types for various complexity levels. In each of the plot the x-axis indicates the experiment number (for example considering the Figure 6.1 below, for experiment run 1, the puzzle generation time is 0.4 seconds and puzzle solution time is 3 seconds) and the y-axis indicates time in seconds.



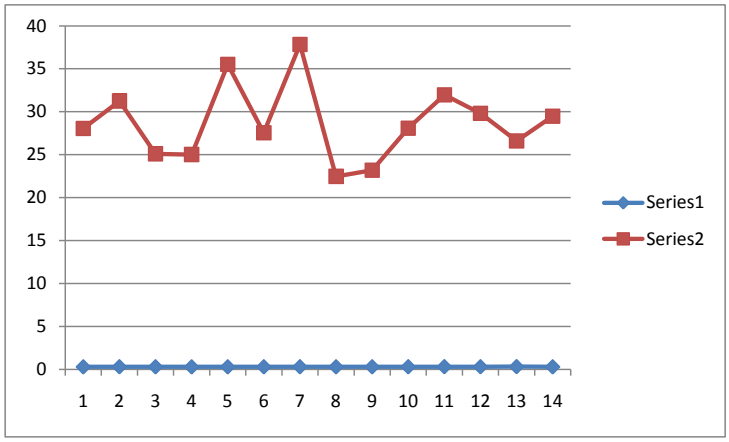
Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.1: Puzzle type 1: Complexity level 1



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.2: Puzzle type 1: Complexity level 2



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.3: Puzzle type 1: Complexity level 3

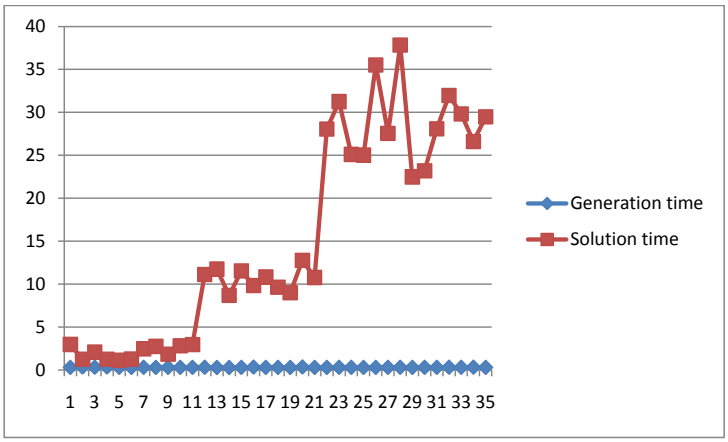
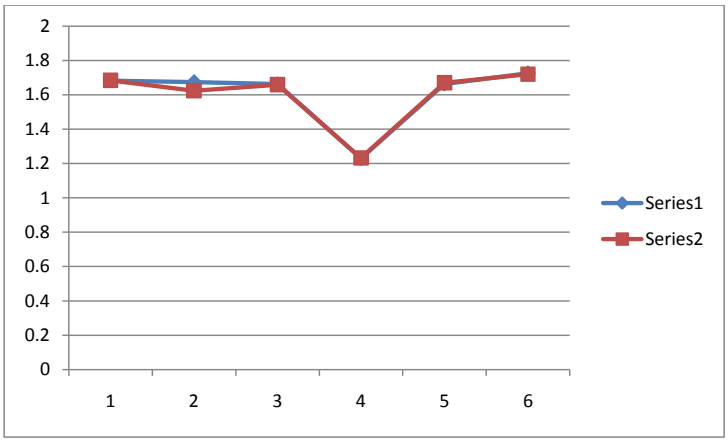
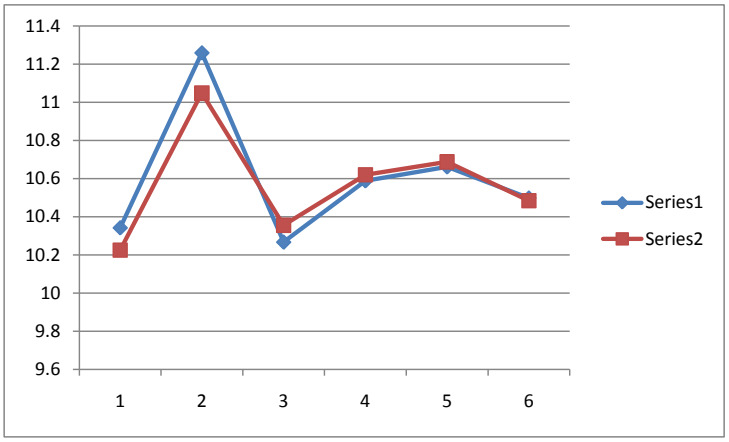


Figure 6.4: Graph summarizing puzzle type 1



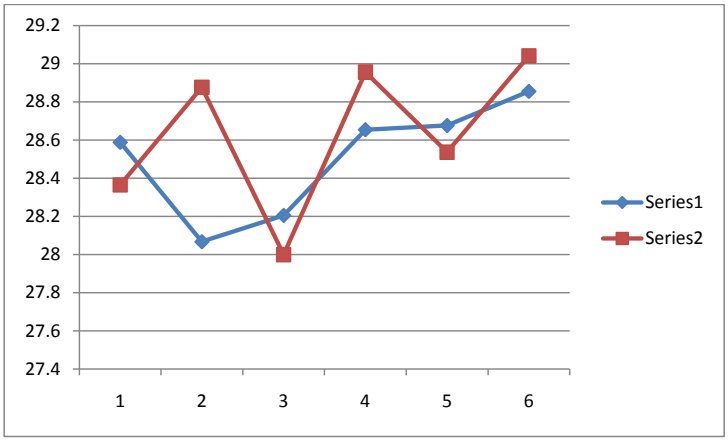
Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.5: Puzzle type 2: Complexity level 1



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.6: Puzzle type 2: Complexity level 2



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.7: Puzzle type 2: Complexity level 3

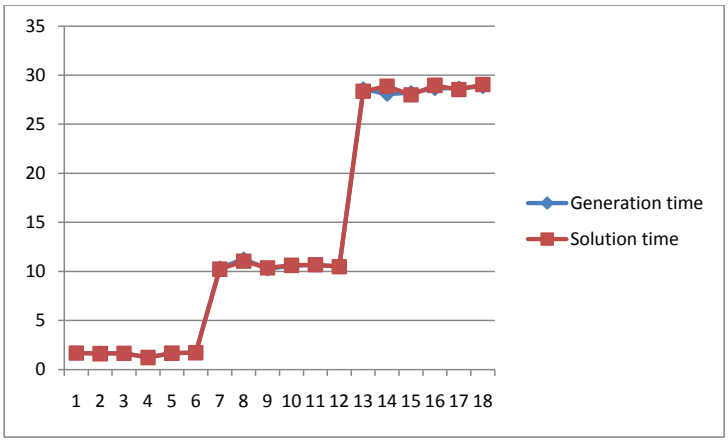
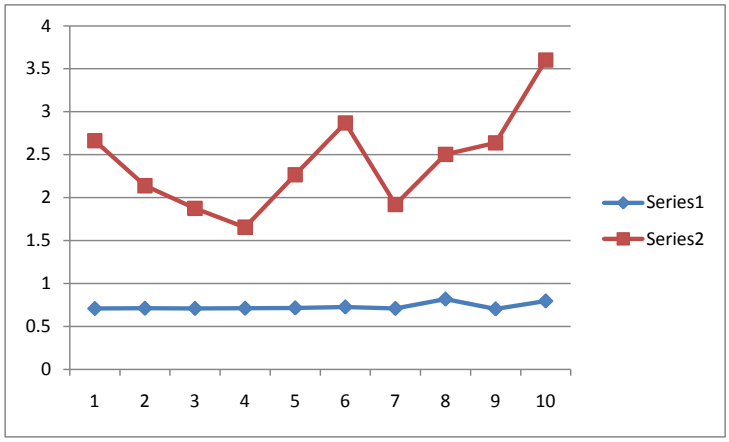
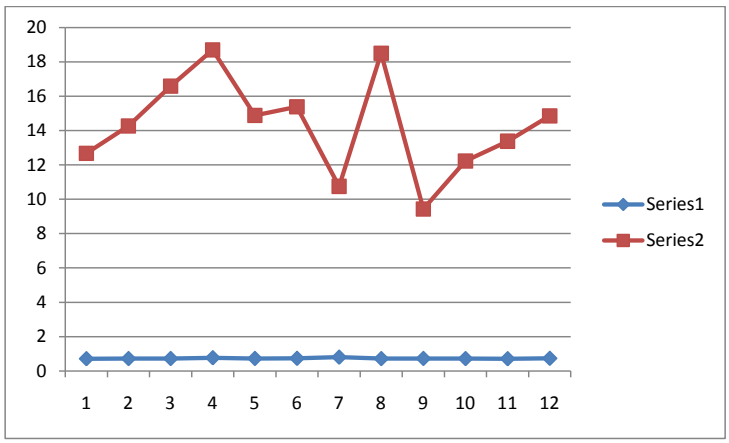


Figure 6.8: Graph summarizing puzzle type 2



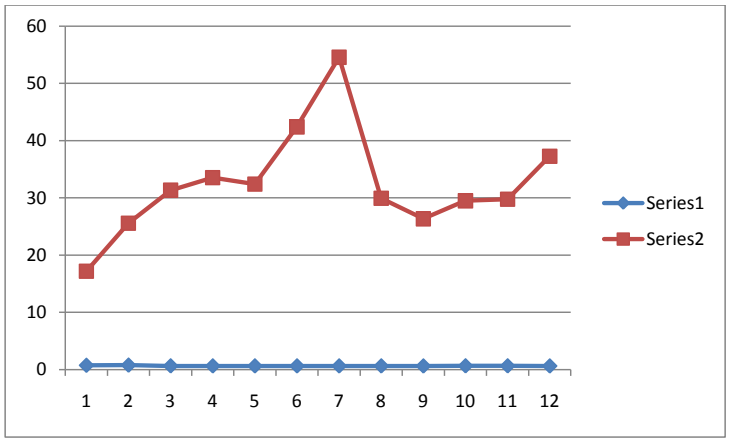
Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.9: Puzzle type 3: Complexity level 1



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.10: Puzzle type 3: Complexity level 2



Series 1: Puzzle generation time, Series 2: Puzzle solution time

Figure 6.11: Puzzle type 3: Complexity level 3

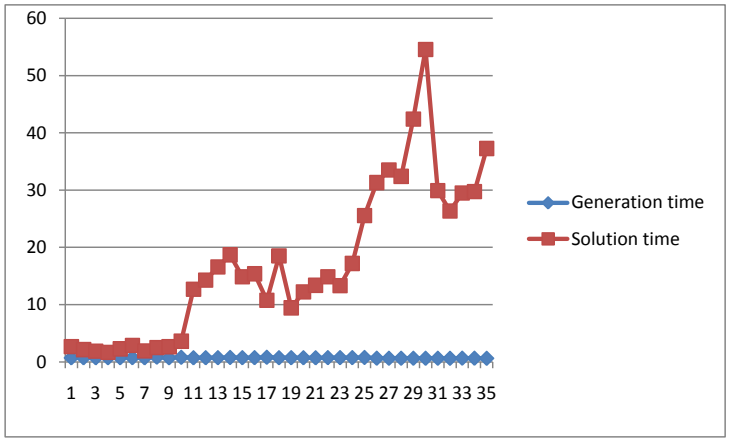


Figure 6.12: Graph summarizing puzzle type 3

6.4 Observation

1. In case of a homogeneous sensor network, if a malicious node assumes identity of multiple nodes, a Sybil attack is always detected.

2. In case of a heterogeneous sensor network, if a malicious node assumes identity of 'q' nodes and solves the puzzles sequentially, and replies to a puzzle as soon as it solves the puzzle, it will escape the detection protocol with probability 1/q!. This is as expected, as explained in mathematical analysis in Case 1 and Case 3.

3. In case of a heterogeneous sensor network, if a malicious node (with computation speed of 'p' times the computation speed of the slowest node in the network and masquerading 'q' nodes) solves the puzzles in parallel and replies to a puzzle at the end of an epoch, it will escape the detection procedure only if q<=p (assuming that the malicious node correctly guesses some parameters not known to any node). However, a Sybil attack will be detected if q>p. This is as expected in case 5.

4. In case of a heterogeneous sensor network, if a malicious node replies to all the puzzles after completely solving all of them, it will not be able to escape the Sybil detection procedure, in case it solves the puzzles either sequentially or in parallel (as explained in case 2 and 4).

6.5 Conclusions and Future work

6.5.1 Conclusion

1. We propose a technique to mitigate Sybil attack in Wireless sensor networks using a combination of computational puzzles and associating a node's identity with a unique

identity key value.

2. We show that in case of a heterogeneous sensor network where nodes can have different computation capability, a compromised node can assume less than or equal to 'p' identities ('p' is the ratio of the computation speed of the compromised node to computation speed of the slowest node in network) if the attacker,

   (a) Knows the identity key value of 'p' legitimate nodes.

   (b) Compromises one of the faster node in the network and correctly guesses its computation speed relative to the slowest node in the network.

   (c) Correctly guesses the epoch interval E.

3. If a compromised node is connected to an external server of infinite computation power, it can only masquerade a bounded number of nodes. This bound is the number of identity key values of legitimate nodes known to the compromised node.

4. Our scheme can be used to detect a Sybil attack in an adhoc network of smart phones while preserving privacy of smart phone users.

6.5.2 Future work

1. If the number of neighbors of a node (N) is very large, then the time required to solve the puzzle of highest complexity level (p^NT) would be very high and a sensor node would spend a lot of its battery computing solutions to a puzzle. Choosing 'M' most suspicious nodes out of N (M< N) to be challenged during Sybil detection is potential future work.

2. A Sybil attack is also possible in VANETs. The unique identifier for vehicles can be used as an identity proof, but it would give away the privacy of vehicles. Applicability of Sybil detection protocol considering mobility is potential future work.

3. Developing a more robust synchronization protocol is also a potential future work.

4. Investigation of other attacks on the Sybil detection proctocol and its mitigation would also be a potential future work. Example of attacks could be,

   (a) A malicious verifier node issues puzzles that do not have a valid solution.

   (b) A malicious node reports wrong information about legitimate nodes to the base station.

   (c) A malicious node frequently issues harder puzzles to legitimate nodes.

REFERENCES

[1] Sybil attack, wikipedia.

[2] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. Dos-resistant authentication with client puzzles. In *Lecture Notes in Computer Science*, pages 170–177. Springer-Verlag, 2000.

[3] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. pages 213–229. Springer-Verlag, 2001.

[4] Murat Demirbas and Youngwhan Song. An rssi-based scheme for sybil attack detection in wireless sensor networks. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, pages 564–570, Washington, DC, USA, 2006. IEEE Computer Society.

[5] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.

[6] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram Khalili. A pairwise key predistribution scheme for wireless sensor networks. *ACM Trans. Inf. Syst. Secur.*, 8:228–258, May 2005.

[7] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. pages 139–147. Springer-Verlag, 1992.

[8] Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[9] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 113 – 127, may 2003.

[10] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 52–61, New York, NY, USA, 2003. ACM.

[11] S. Misra and S. Myneni. On identifying power control performing sybil nodes in wireless sensor networks using rssi. In *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, pages 1 –5, dec. 2010.

[12] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis defenses. In *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, pages 259 – 268, april 2004.

[13] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. Spins: Security protocols for sensor networks. *Wireless Networks*, pages 521–534, 2002.

[14] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta. Limiting sybil attacks in structured p2p networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2596 –2600, may 2007.

[15] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 272 – 282, may 2004.

[16] Jiangtao Wang, Geng Yang, Yuan Sun, and Shengshou Chen. Sybil attack detection based on rssi for wireless sensor network. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 2684 –2687, sept. 2007.

[17] Brent Waters, John A. Halderman, Ari Juels, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance, 2004.

[18] Ren Xiu-li and Yang Wei. Method of detecting the sybil attack based on ranging in wireless sensor network. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pages 1 –4, sept. 2009.

[19] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 267–278, New York, NY, USA, 2006. ACM.

[20] Qinghua Zhang, Pan Wang, D.S. Reeves, and Peng Ning. Defending against sybil attacks in sensor networks. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 185 – 191, june 2005.

[21] Tong Zhou, R.R. Choudhury, Peng Ning, and K. Chakrabarty. Privacy-preserving detection of sybil attacks in vehicular ad hoc networks. In *Mobile and Ubiquitous Systems: Networking Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on*, pages 1 –8, aug. 2007.