Topology-based network security

Tiit Pikma
Supervised by Vitaly Skachek

Research Seminar in Cryptography
University of Tartu, Spring 2013

1 Introduction

In both wired and wireless networks, there is the possibility of a wiretapper listening on the communication between nodes in the graph. There are many cryptographic protocols that secure us against this, but Kamal Jain proposes an alternate solution to this problem[1]. The suggested protocol's security is based on the network topology and all relevant nodes in the graph working together. Furthermore, it assumes no shared secret between the sender and the receiver.

Jain based his work off of Cai and Yeung's[2] who proposed a wiretapping model in the setting of broadcasting messages in acyclic graphs. They gave a very tight sufficient condition to ensure that the message could not be wiretapped. Jain widens the condition to also become necessary. His approach also works on cyclic graphs and is used for unicasting.

The aim of this report is to give a longer and perhaps more understandable explanation of the protocol, including an example on a sample graph. Both of these were lacking in the short original paper by Jain. We also give out some ideas to expand on this protocol, to make it work on longer than one-bit messages.

In Section 2 we give the model along with the assumptions for the problem we are trying to solve. In Section 3 we explain the protocol proposed by Jain, give an example, and prove that this protocol works and is secure. In Section 4 we sketch some ideas which would allow us to send longer messages than proposed by Jain and also give a short example network. Finally, in Section 5 we outline our conclusions.

2 Description of the problem

Let us have a network represented by a directed graph $G = (V, E)$, where V is the set of vertices and E the set of edges. Each node in this network has a random number generator. A special node, the sender s, wants to send a

message to another special node, the receiver r[1]. The sender and receiver have no shared secret. There is a wiretapper who can read all messages on a set of edges. The set of edges the wiretapper can listen on is one from a given collection of sets, $\mathcal{A} = \{A_1, A_2, ..., A_n\}$, where each $A_i \subseteq E$. Obviously the sender does not want the wiretapper to get any information about the message being sent to the receiver.

Given G and $\mathcal{A}$ we want to ascertain whether the sender can send the message without leaking any information. We assume that all nodes in the graph cooperate towards this goal. We also assume that all edges are reliable (no lost packets) and perfectly synchronized (packets on a link arrive in the order they were sent in). Clearly, if there is no path from s to r in G, then no message can be sent, regardless if it is being wiretapped or not. So we assume there is a path from the sender to the receiver.

Let $S \subset V$ be any set of nodes which contains the sender but not the receiver. We notate $\delta_o(S)$ as the set of edges going from S to $\overline{S}$ (the complement of S). Analogously we notate $\delta_i(S)$ as the set of edges going from $\overline{S}$ to S. Finally, let $\delta(S) = \delta_o(S) \cup \delta_i(S)$.

For simplicity let us assume that the messages being sent are binary strings of length 1.

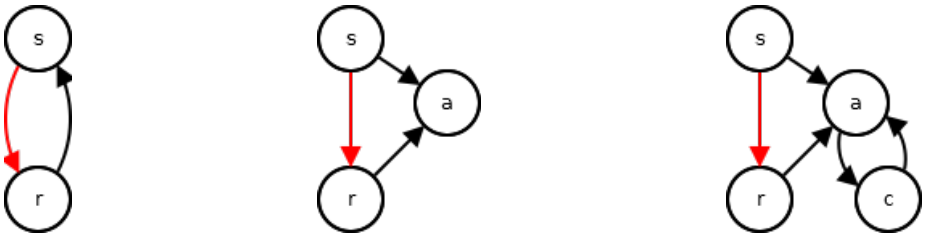
3 Proposed solution

In this section we show a sufficient and a necessary condition, followed by a sufficient and necessary condition derived from the previous two. Given that this condition is satisfied, we construct an algorithmic protocol which provides a solution to the problem described in Section 2.

3.1 Conditions for the solution

The first condition to consider is that $\delta_o(S) \not\subseteq A$ for any S and any $A \in \mathcal{A}$. Or in other words, for all sets of nodes S, where the sender is in S and the receiver is not, there exists an edge going from S to $\overline{S}$ which isn't wiretapped. Let's call this condition $C1$. $C1$ implies that we can send a message without leaking any information, because there exists a secure path from s to r. While this condition is necessary for acyclic graphs (otherwise the wiretapper knows everything that is sent from s to r and, since there is no shared secret, knows everything the receiver does), it is not necessary for graphs with cycles. Consider a graph with only two nodes, s and r, and two edges, (s, r) and (r, s). Even if the wiretapper is listening on (s, r) (see Figure 3.1a), i.e. $\delta_o(S) = A$, a message can still be sent: r sends a random bit to s via (r, s), s adds it to the message to be sent (modulo 2) and sends this to r via (s, r). The receiver can subtract the bit it sent to s from this string to recover the message, while the wiretapper only

[1] This network can be expanded to support multicasting, i.e. have more than one receiver, but we look at a single receiver for simplicity's sake.



(a) (b) (c)

Figure 3.1: Graphs demonstrating counter-examples for the different conditions. Wiretapped edges are highlighted in red.

sees a random bit. Therefore this first condition is sufficient, but not always necessary.

Let's widen this condition to $\delta(S) \not\subseteq A$ for any S and any $A \in \mathcal{A}$. Or once again in other words, for all sets of nodes S, where the sender is in S and the receiver is not, there exists either an edge going from S to $\overline{S}$, or an edge going from $\overline{S}$ to S which isn't wiretapped. Let's call this condition $C2$. $C2$ is necessary to send a message without leaking information: if the wiretapper is listening on a cut of the graph with s and r on different sides (i.e. all paths going from s to r and from r to s have a wiretapped edge), then the wiretapper knows everything they send each other and no secure message can be sent. Yet this condition is not sufficient.

Consider a graph with three nodes, s, r, and a, and three edges, (s, r), (r, a), and (s, a). Suppose only (s, r) is wiretapped, i.e. $\delta(\{s\}) \not\subseteq A$ as $(s, a) \notin A$ and $\delta(\{s, a\}) \not\subseteq A$ as $(r, a) \notin A$ (see Figure 3.1b): the condition is satisfied, but nothing can be sent to the receiver without leaking it. Therefore the condition is no longer sufficient.

The problem with $C2$ was that the node a had no outgoing edge—so in some sense it was irrelevant. But having no outgoing edges is not the only situation when the condition fails to be sufficient: we can extend the counter-example to have another node c with the extra edges (a, c) and (c, a). Now all nodes have outgoing edges, but a and c are still irrelevant to our transmission from s to r (see Figure 3.1c). From this we can observe that only nodes that have a path to r are relevant parts of our network; everything else is unnecessary for that transmission.

Removing all irrelevant nodes for a transmission is a key pre-processing step that makes $C2$ sufficient and necessary.

3.2 The proposed algorithmic protocol

As briefly explained, condition $C2$ in Section 3.1 is necessary to send a message without leaking any information. Now we show that if we do the aforementioned preprocessing, then it becomes also sufficient: we do this by outlining

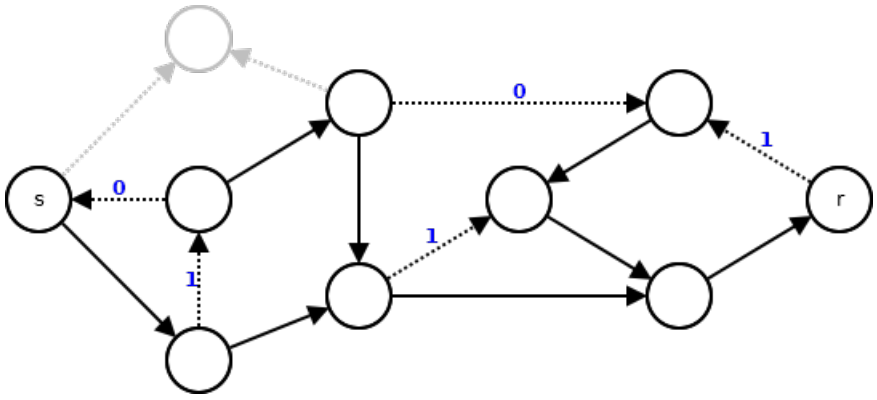


Figure 3.2: An example graph. Irrelevant parts are denoted in gray, edges in H are denoted in solid black, edges in G but not in H are denoted with dotted lines. Randomly generated messages on edges are in blue.

an algorithmic protocol to send a message from the sender to the receiver and prove that this does not leak any information to the wiretapper.

Given a graph, first delete all the irrelevant parts of it (as defined in Section 3.1) and denote this pruned graph G[2]. Next, pick a spanning arborescence H on the graph: a spanning arborescence is like a spanning tree, except all edges are directed towards the root (see Figure 3.2). This means that there is exactly one path from any node to the receiver in H. Since we pruned G to not include any nodes which do not have paths to r, H includes every node in G—it does not however have to contain every edge, as some node in G might contain more than one path to r. We can find H by reversing all the edges in G (now there is at least one path from r to every node in H) and doing a breadth-first traversal of G starting from r. After this, we reverse all the edges back to their original direction[3]. So we know at least one H exists[4] and we know how to find it.

On every edge (u, v) that is in G but not in H, we ask u to send a random message to v; now all edges except the ones in the spanning arborescence H have a value to send (Figure 3.2). Note that every leaf in H has only one edge on which it has nothing to send yet—the edge going to its parent—as all others send random data from the previous step. Pick a random leaf and denote this edge (u, v). There are two possible cases (see Figure 3.3 for examples of both):

u is not the sender We take the sum (modulo 2) of the messages u receives on all incoming edges and sends on all outgoing edges except for (u, v).

[2]Although before we denoted the first graph also G, we redefine this to the pruned graph for convenience's sake, as we will not be using the original graph at all.

[3]Note that if we do a breadth-first search starting from r in the original, non-pruned graph, then we can only reach nodes which are relevant to the graph. So during implementation we can combine pruning and finding H into one step. For the purposes of this paper, let's keep them separate for simplicity.

[4]There can be multiple spanning arborescences.

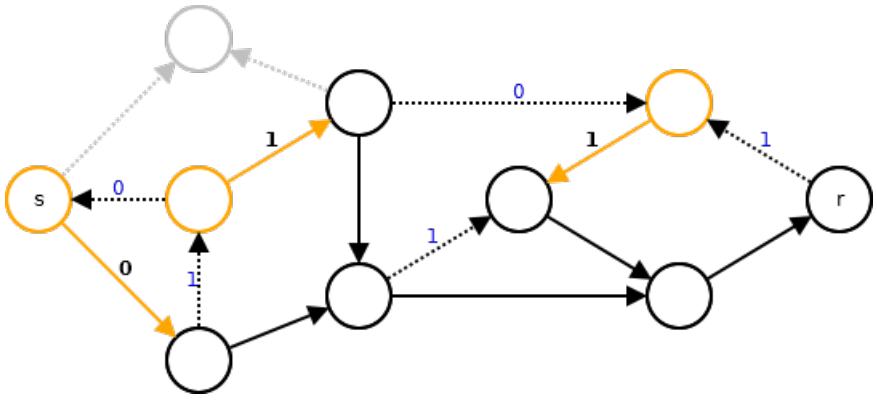


Figure 3.3: An example of calculating messages to send up the arborescence. The leaves of the arborescence and edges to their parents are highlighted in orange. The calculated messages are in bold black. The message bit being sent is 0.

We send this sum on the edge (u, v). This ensures that the sum (modulo 2) of all incoming edges is equal to the sum (modulo 2) of all outgoing edges; or in other words, the sum of all edges (modulo 2) is zero.

u is the sender We take the sum (modulo 2) of the messages u receives on all incoming edges, the messages it send on all outgoing edges except for (u, v), and the message to be sent. We send this sum on the edge (u, v). This ensures that the sum (modulo 2) of all the incoming edges with all of the outgoing edges is the message.

After we have done this for all the leaves, we can momentarily exclude them from H: now all the leaves in this graph have only one edge on which they have nothing to send yet. We repeat this process until we reach the receiver and all edges have something to send on them. See Figure 3.4 for the completed example.

Lemma 1. *For any S, the sum (modulo 2) of the messages being sent on $\delta(S)$ is the message being sent from s to r.*

Proof. We prove this via induction.

Base: If $S = \{s\}$, then the lemma holds by construction.

Induction step: Let the lemma hold for S. Let u be any node not in S and not r. We want to prove the lemma for $S \cup \{u\}$.

Recall that we constructed the messages such that the sum (modulo 2) of messages over $\delta(\{u\})$ is zero: because of this, the sum (modulo 2) of messages over $\delta(S \cup \{u\})$ is the same as the sum (modulo 2) of messages over $\delta(S)$, so the lemma holds for $S \cup \{u\}$. □

Theorem 2. *The node r receives the message s sent.*

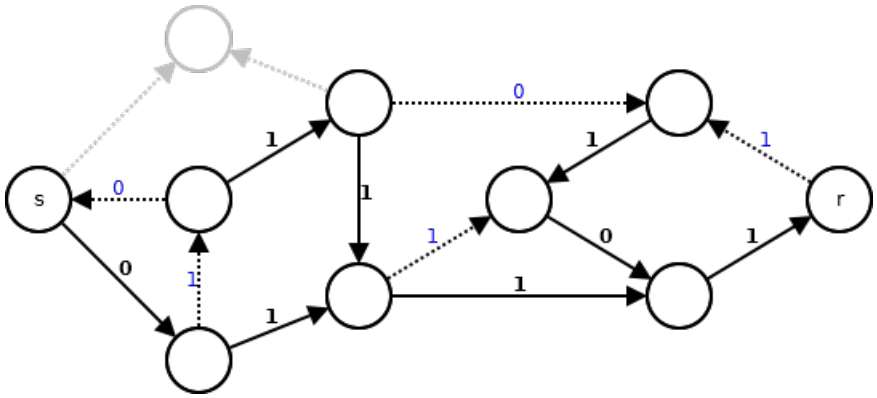


Figure 3.4: An example of a graph where the message bit 0 is being sent.

Proof. Let S contain every node in G except for r. Then $\delta(S) = \delta(\{r\})$, because they both are the set of all edges incident to r. According to Lemma 1, if r sums up (modulo 2) all the messages it sends and receives, the result will be the message being sent from s to r. □

We proved that using this protocol, r does in fact receive the message sent from s. Now we need to show it is secure.

First off, we can see that pruning the graph had no effect on security: edges which are not in G do not carry any messages, so any attackers are welcome to tap them as much as they want.

Let's call the assignment of messages to all edges in a pruned graph a *configuration*. A configuration is valid in this protocol if for every node except for s and r, the sum (modulo 2) of messages on the edges incident to that node is zero, and given a fixed message b, the sums of messages on the edges incident to s and r are b.

Lemma 3. *Pick any spanning arborescence of the graph. Fix any value for the message. Any assignment of messages to all edges not in the spanning arborescence can be extended into a valid configuration of the graph.*

Proof. We have messages for all edges not in the arborescence and none for edges in it. We can use the same recursive approach as above to assign messages to all edges in the arborescence from leaves to root, so we end up with a valid configuration by construction, with the validity of edges incident to r proven in Theorem 2. □

For the following, we can ignore directions in G, because the direction a message is going on a tapped edge does not matter. Let the graph have n nodes and m edges. Since there is one edge for each node in H except for r (the one going to the node's parent), then there are $n - 1$ edges in H. Therefore there are $m - n + 1$ edges that are in G and not in H.

Corollary 4. *There are 2^{m-n+1} valid configurations for a given message bit.*

Proof. There are $m-n+1$ non-tree edges in the undirected G. Any assignment of messages to non-tree edges can be extended into a valid configuration (as per Lemma 3). □

Recall from before that we sent a random message on all edges in G and not in H, so during this protocol we have to generate $m-n+1$ random messages.

Corollary 5. *Given a fixed message, each valid configuration is equally likely.*

Proof. We can map each unique set of random numbers to a unique valid configuration for the fixed message, so if the random numbers are uniformly distributed, so are the configurations. □

Lemma 6. *If all edges except for those of a spanning tree are tapped, the tapper does not get any information about the message.*

Proof. Messages assigned to non-tree edges are completely random and are independent of the message being sent from s to r. Furthermore, any assigment of messages to non-tree edges can be extended into a valid configuration for all possible values of the message. Therefore tapping the messages on those edges does not leak any information about the message beint sent from s to r. □

Lemma 7. *If all edges except for those of an undirected path between s and r are tapped, the tapper does not get any information about the message.*

Proof. Still ignoring direction in G, choose an undirected path P from s to r. A path is a tree, so we can use an unweighted version of Prim's algorihm[3] to expand P into a spanning tree on the unweighted G. Intuitively, what we are doing is the reverse of the recursive algorithm from above: instead of removing leaves from H, we are adding them to P, at each step computing the value on the edge in the spanning tree.

Let N be the set of nodes in P. Note that the sum of messages over $\delta(N)$ is zero because N contains both s and r which cancel each other out. While there exists a node in G but not in N, pick any edge (u, v) in $\delta(N)$ (i.e. an edge from N to $\overline{N}$, where u is in N and v in $\overline{N}$), add it to the spanning tree, and add the newly spanned node v to N. The value for any edge (u, v) picked this way can be computed from the values of all other edges incident to v: since v is not the sender nor the receiver—which are already in N—, the value of (u, v) needs to be such that the sum (modulo 2) of all edges has to be zero. Because the value of this edge can be calculated in this way, there is no need nor point for the attacker to wiretap this (see Figure 3.5). Once we have extended P to a spanning tree, we can apply Lemma 6 to show that the tapper gets no information. □

Theorem 8. *If for any S there is at least one edge in $\delta(S)$ that is not tapped, i.e. $\delta(S) \not\subseteq A$ for any S and any $A \in \mathcal{A}$, the tapper does not get any information about the message.*

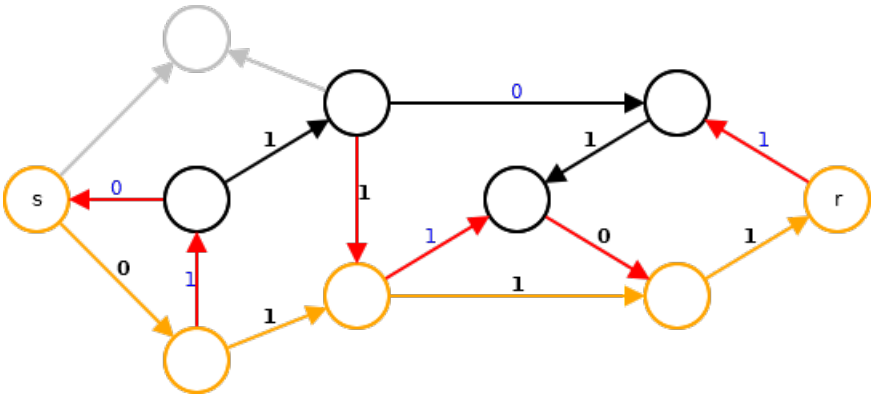


Figure 3.5: The path P is in orange. There is no gain in tapping any of the red edges, because the message sent on them can be calculated using the other edges incident to the non-orange node. So we can add one of the red edges to the untapped orange spanning tree.

Proof. For all S, let each edge in $\delta(S)$, i.e. each edge in the cut, have capacity 1 if it is not in any $A \in \mathcal{A}$, and capacity 0 otherwise. So an edge will have positive capacity if we can send a message through it without being tapped, and zero capacity if we can't. The condition of the theorem states that there exists at least one edge with capacity 1 in all such cuts in the graph. According to the max-flow min-cut theorem[4], the minimum capacity over all s-r cuts is equal to the maximum value of an s-r flow. So the maximum flow in this graph is at least 1, meaning that there exists an undirected path from s to r with flow at least 1, i.e. that is not tapped. The thorem follows from Lemma 7. □

Theorem 8 proves that the protocol described in Subsection 3.2 is secure.

4 Sending longer messages per round

Previously we only looked at messages that are binary strings of length one. Regardless, we stayed relatively general in the proofs, so two ideas to send longer messages follow. The following ideas are those of the author of this paper and are not taken from Jain's article.

First off, we could just send longer bit strings and when calculating the sums of messages over incident edges, we don't do it modulo 2, but modulo 2^l, where l is the bit-length of the message. All edges that are not in the spanning arborescence are assigned messages uniformly and randomly from $\left[0, 2^{l-1}\right]$. For all nodes that are not the sender nor the receiver, we still construct the messages such, that the sum (modulo 2^l) of incident messages is zero. This way the induction proof in Lemma 1 still holds and we can send the message. The security of this is sadly not so easily translatable, as for example in Lemma 7 we used the fact that the starting sum of messages over edges in $\delta(N)$ would be

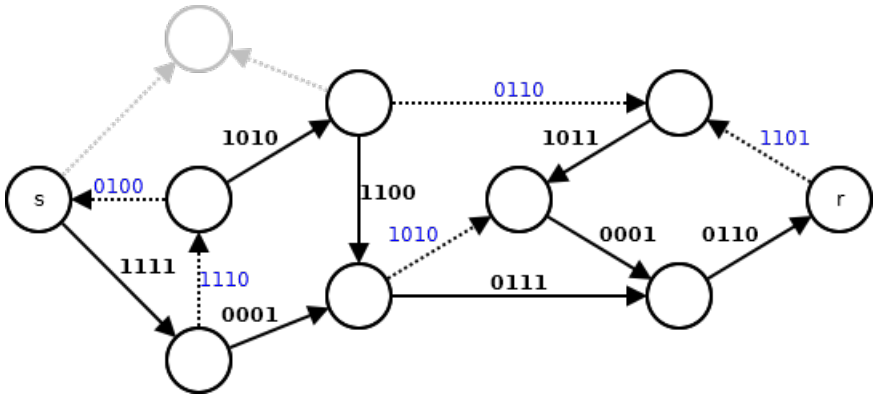


Figure 4.1: An example configuration using the XOR approach to send the message 1011.

zero, because s and r would cancel each other out—this does not hold if we have longer messages (e.g. if the message is 1 and l is 2, then the sum of messages on the edges incident to s and r would be 2, not 0), so this step in the proof would have to be redone.

Alternatively, we can look at the protocol in a more intuitive way. The main idea is that for all S in the graph, the message is distributed across all the edges in the cut $\delta(S)$, so if the wiretapper misses at least one edge in all of the cuts, then no information is leaked about the message. Previously we did this distribution by adding together bits modulo 2: if any bit is missing, then we don't get the sum. Another way to do this distribution would be with the XOR operation: if we are missing any messages from a cut, then we can't XOR them together to get the message being sent from s to r. This also has the nice property that XOR-ing the message with itself yields zero (unlike with the previous idea). The algorithm of the protocol would remain the same: we assign random messages of length l to all non-arborescence edges, and start computing messages for the only remaining edges in leaf nodes by either making sure the XOR is zero, if the leaf is not the sender, or that the XOR is the message, if the leaf is the sender. See Figure 4.1 for an example configuration.

5 Conclusion

As shown in Section 3.2, this protocol works and is secure. We also showed that precondition for it is sufficient and necessary, so when a wiretapper does in fact have all edges in a cut tapped, there is nothing we can do against that; at least not when using this approach to security.

One problem with the current solution is that we have to use almost the entire network, excluding the irrelevant parts, to send one message between two nodes. This presents us with a scalability problem as it is fair to assume

that other nodes in the network also wish to communicate: using the current approach only one pair can do so at a time. This, and the ability to do multicasting, i.e. have one sender communicate with more than one receiver at once, are things that need to be improved in the protocol detailed in this report[5].

References

[1] Jain, K., "Security based on network topology against the wiretapping attack". Wireless Communications, IEEE , vol. 11, no. 1, pp. 68-71, Feb 2004.

[2] Cai, N, Yeung, R. W., "Secure network coding". Proceedings of IEEE International Symposium on Information Theory 2002, p. 323, 2002.

[3] Prim, R. C., "Shortest connection networks and some generalizations". Bell System Technical Journal, 36, pp. 1389-1401, 1957.

[4] Ford, L. R., Fulkerson, D. R., "Maximal flow through a network". Canadian Journal of Mathematics 8, pp. 399-404, 1956.

[5] Jain[1] did actually propose an idea of using one-way functions to support multicasting but we did not look at that in this paper.