Reliable Communication in Hybrid Authentication and Trust Models

Rowdy Chotkan
Delft University of Technology, The Netherlands

Bart Cox
Delft University of Technology, The Netherlands

Vincent Rahli
University of Birmingham, UK

Jérémie Decouchant
Delft University of Technology, The Netherlands

Abstract

Reliable communication is a fundamental distributed communication abstraction that allows any two nodes within a network to communicate with each other. It is necessary for more powerful communication primitives, such as broadcast and consensus. Using different authentication models, two classical protocols implement reliable communication in unknown and sufficiently connected networks. In the former, network links are authenticated, and processes rely on dissemination paths to authenticate messages. In the latter, processes generate digital signatures that are flooded throughout the network. This work considers the hybrid system model that combines authenticated links and authenticated processes. Additionally, we aim to leverage the possible presence of trusted nodes (e.g., network gateways) and trusted components (e.g., Intel SGX enclaves). We first extend the two classical reliable communication protocols to leverage trusted nodes. Then we propose `DualRC`, our most generic algorithm that considers the hybrid authentication model by manipulating dissemination paths and digital signatures, and leverages the possible presence of trusted nodes and trusted components. We describe and prove methods that establish whether our algorithms implement reliable communication on a given network.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computer systems organization → Fault-tolerant network topologies

Keywords and phrases Reliable communication, Byzantine, Authentication models, Trust

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.25

Related Version *Full Version*: `https://arxiv.org/abs/2408.08060`

Funding *Rowdy Chotkan*: This work was partially supported by NWO/TKI grant BLOCK.2019.004.
Vincent Rahli: This work was partially supported by the Engineering and Physical Sciences Research Council (EPSRC) grant number EP/W034514/1.

1 Introduction

Distributed systems involve autonomous computing processes (also called nodes) that communicate using a network to perform a cooperative task. Formalization efforts have led to the definition of distributed computing abstractions defined by a set of properties [13]. In particular, fault-tolerant distributed computing abstractions tolerate a limited number of faulty nodes. The most general fault model is the Byzantine model, which allows processes to deviate from a specified protocol in unrestricted ways, making such systems well-suited to handle real-world scenarios. For example, protocols that implement the Byzantine consensus abstraction are involved in Blockchain systems [15, 43].

Table 1 Reliable communication protocols, their system models, and correctness conditions. The three detached bottom rows are novel protocols discussed in this paper.

Protocol	**Auth. links**	**Auth. nodes**	**Trust. nodes**	**Trust. comp**	**Correctness condition**
DolevU [16]	✓	✗	✗	N/A	$2f+1$ vertex-connectivity
SigFlood [17]	✗	✓	✗	✗	$f+1$ vertex-connectivity
DolevU-T (§3, Alg. 1)	✓	✗	✓	N/A	`DolevU-T-Verif` (§3.3.1 and §3.3.2, Alg. 5 or Alg. 6)
SigFlood-T (§4, Alg. 2)	✗	✓	✓	✓	`SigFlood-T-Verif` (§4.2, Alg. 5 or Alg. 6 with $2f+1$ replaced by $f+1$)
DualRC (§5, Alg. 3)	✓	✓	✓	✓	`DualRC-Verif` (§5)

Distributed computing abstractions can be built on top of each other, which simplifies implementations and allows for simpler proofs of correctness. Arguably, the simplest distributed computing abstraction is *reliable communication (RC)* [16, 27, 35] because it allows two processes to authenticate each other's messages. In an unknown and incomplete network, reliable communication relies on broadcast and requires that: (i) when a correct process broadcasts a message, then the message is authenticated (or delivered) by all correct nodes; and (ii) when a message originating from a correct process is delivered, it was indeed sent by this correct process. Reliable communication is closely related to *reliable broadcast* [12] and ensures the same properties when the original sender is correct. As a consequence, reliable communication is sometimes referred to as reliable broadcast with an honest dealer.

State-of-the-art reliable communication protocols have either considered that all nodes are authenticated (i.e., can generate and verify signatures) or, instead, that all nodes rely on authenticated links [16]. However, with the recent development of complex systems that involve a large variety of nodes – some using digital signatures and some using point-to-point authenticated communication – there is a need to revisit the reliable communication abstraction and consider hybrid models.

Several academic works have explored hybrid process models and additional trust assumptions, including trusted nodes [39, 40] and trusted components [14, 15, 25, 42], primarily in the context of higher-level distributed computing abstractions such as consensus. The impact of trusted nodes and trusted components on lower-level distributed computing abstractions, such as reliable communication, has not been evaluated. In this work, we revisit state-of-the-art reliable communication protocols and extend them to tolerate the simultaneous presence of both authenticated and non-authenticated nodes, while also leveraging the possible presence of trusted nodes or nodes equipped with a trusted component. Tab. 1 lists the state-of-the-art RC protocols that we consider in this paper, along with the three novel protocols that we introduce, and compares their system models and correctness conditions.

As a summary, this work makes the following contributions:

- We extend the state-of-the-art reliable communication protocols to leverage trusted nodes in unknown networks under the global fault model. First, we modify Dolev's [16] reliable communication protocol, which disseminates messages across authenticated links along with their propagation paths, facilitating message authentication (§3). Second, we modify the flooding-based reliable communication protocol [17] – referred to as `SigFlood`. This protocol, which relies on authenticated nodes, ensures that the dissemination of a message is accompanied by the digital signature of the original sender (§4).
- We present `DualRC`: the first reliable communication protocol that supports the presence of both authenticated and non-authenticated processes, and leverages trusted nodes and trusted components (§5). This protocol disseminates messages that contain dissemination

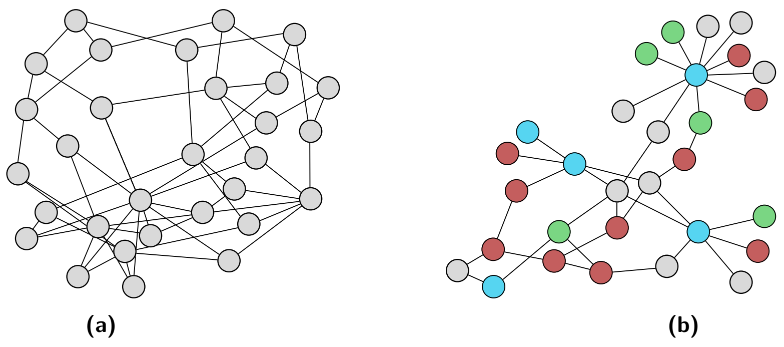

Figure 1 (a) In a network of 30 authenticated nodes with connectivity (3), both SigFlood and `DualRC` ensure reliable communication when $f = 1$. (b) In a complex network with 10 authenticated (red), 10 non-authenticated (gray), 5 authenticated trusted (green), and 5 non-authenticated trusted (blue) nodes, only `DualRC` achieves reliable communication.

paths, signed payloads, and signed dissemination paths. `DualRC` aims at supporting the largest possible range of network topologies where all authenticated and non-authenticated nodes can communicate reliably. Interestingly, `DualRC` leverages the presence of trusted components to translate signed dissemination paths into trusted signatures that vouch for the authenticity of a message. Fig. 1 shows two networks to illustrate that `DualRC` implements reliable communication in more generic networks than previous algorithms.

- Finally, for each of our protocols, we establish the necessary and sufficient conditions that the network topology must meet to enforce all reliable communication properties. Additionally, we present algorithms that allow for the verification of these conditions and evaluate their complexity.

2 Models and Problem Statement

Network. We consider a static set $V=\{p_1, p_2, \ldots, p_N\}$ of N processes, which are assumed to know the system size N. Processes are interconnected by a communication network, depicted as an undirected graph $G=(V, E)$ where each node $p_i \in V$ corresponds to a process, and each edge $e_{ij} \in E$ denotes a unicast communication channel between two processes. This paper uses the terms "process" and "node" interchangeably, treating a process as equivalent to the network process that hosts it. Furthermore, we assume that the network topology is not known to nodes and remains static. However, processes are aware of their direct neighbors and their identities. Two nodes that are not directly connected by an edge $e_{ij} \in E$ must rely on other, potentially Byzantine, nodes to relay their messages. Communication channels are authenticated, ensuring that only the two nodes at their endpoints can use them for message transmission. The channels are also reliable, meaning that messages are not altered or lost during transmission. Finally, the channels are asynchronous, allowing for unbounded transmission delays. Tab. 2 summarizes the graph-related notations we use in this paper.

Channels interface. We assume that processes have access to a virtual interface al[1] that abstracts their communication channels. The virtual interface al exposes two functions: `Send` and `Receive`. A node p_i can send a message m to a neighbor p_j by calling function $\langle al, \texttt{Send} | p_j, m\rangle$. Similarly, an incoming message m sent by a neighbor p_j triggers the function $\langle al, \texttt{Receive} | p_j, m\rangle$ that node p_i executes.

[1] al is an acronym for *authenticated channel*.

Table 2 Notations and definitions.

Notation	Definition
G	An undirected graph.
$\Gamma_G(u)$	The neighbors of node u in graph G.
$\kappa_G(u,v)$	Vertex-connectivity between two nodes u and v in graph G.
$f(G)$	The graph whose vertices are G's untrusted nodes and whose edges connect the untrusted nodes of G that are neighbors in G or connected by a path of trusted nodes in G (cf. Def. 2).
$g(G,f(G),u)$	The graph $f(G)$ to which node $u \in G$ has been added along with its edges (direct connections with nodes in $f(G)$, or connection to them through a path of trusted nodes in G) (cf. Def. 3).
$\emptyset_p$	An empty path.
$\emptyset_l$	An empty list.
$\sigma_i(m,p_s)$	Signature of process p_i on payload data. m broadcast by process p_s.
$\sigma_i(pa,m,p_s)$	Signature of process p_i on a dissemination path pa, and payload data m broadcast by process p_s.
$\sigma_{TC_i}(m,p_s)$	Signature of process p_i's trusted component on m broadcast by process p_s.
`isTrusted`(p_i)	Utility function that indicates whether process p_i is trusted.
`isTC`$(\sigma_i(m,p_s))$	Utility function that indicates whether a signature $\sigma_i(m,p_s)$ was generated by process p_i's trusted component on payload data m broadcast by p_s.

Objective. We aim to implement reliable communication (RC). RC allows any two correct processes to exchange messages and authenticate them. The interface of an RC protocol includes two functions, $\langle$`RC-broadcast`$|m\rangle$ and $\langle$`RC-deliver`$|m\rangle$, which respectively broadcast a message m and transfer a message m to a higher-level application. In an unknown network, RC is implemented through broadcasting to ensure that a message reaches all nodes, with only the intended recipient delivering the message. We refer to the initial sender of a message as *broadcaster*. Def. 1 defines the RC distributed computing abstraction.

▶ **Definition 1.** *A reliable communication algorithm ensures the following three properties:*

- ***RC-No duplication:*** *A correct process* `RC-delivers` *a message at most once.*
- ***RC-No creation:*** *If a correct process* `RC-delivers` *a message, then it was* `RC-broadcast`*.*
- ***RC-Validity:*** *If a correct process* `RC-broadcasts` *a message, then all correct processes eventually* `RC-deliver` *it.*

Node categories. We divide the set of all nodes, V, into four subsets (V_{auth}, $V_{n.auth}$, V_t, and V_{tc}), as illustrated in Fig. 2. The composition of these subsets is known to all nodes and is defined as follows: V_{auth} contains the authenticated nodes, which are capable of using digital signatures, whereas $V_{n.auth}$ comprises non-authenticated nodes, which cannot. Non-authenticated nodes rely on authenticated links to authenticate the messages they receive and allow others to authenticate their messages. We define $V_t \subseteq V$ as the set of trusted nodes, which are known to be correct, i.e., they never deviate from the protocol specification [1, 7, 34, 39]. However, they are not necessarily authenticated. Finally, we define $V_{tc} \subseteq V$ as the set of nodes that are equipped with a *trusted component* (TC). Note that $V_t \cap V_{tc}$ is not necessarily empty, which means that a trusted node might also be equipped with a trusted component. TCs can store data and execute code in a secluded area inaccessible to their host operating system. However, TCs rely on their host to interact with the network and obtain CPU cycles. TC-augmented nodes are strictly considered authenticated, i.e., $V_{tc} \subset V_{auth}$. This is in line with modern TCs, which are usually authenticated using a remote attestation procedure that employs cryptographic primitives [31].

Fault model. We assume that up to f out of the N nodes are Byzantine. Faulty nodes exclusively belong to $V \backslash V_t$. Such nodes may deviate from protocol specifications in arbitrary ways but cannot break cryptographic primitives. For example, a faulty node might modify or drop messages it is expected to send. When equipped with a TC, a faulty node may not interact with it as intended.

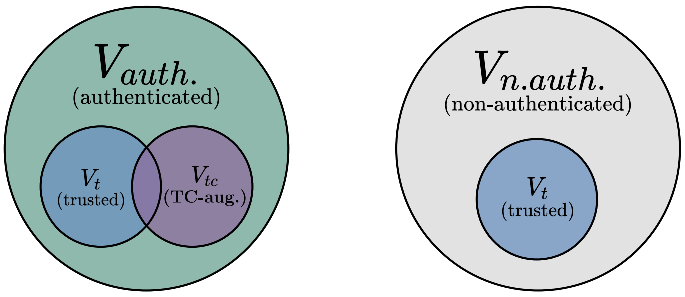

Figure 2 Taxonomy of processes.

Trusted component. In our final algorithm, `DualRC`, a trusted component – which could be implemented with any off-the-shelf trusted execution environment (e.g., Intel SGX) – assumes two functions. First, it can verify a broadcaster's signature and generate a signed message to attest that it authenticated it. Second, it can take $f+1$ signed dissemination paths that relate to a given payload as input, verify that the paths are vertex-disjoint, and then generate a signature to attest that the payload has been authenticated based on signed paths. To conserve space, a formal definition of this trusted component interface is omitted.

3 `DolevU-T`: Reliable Communication in the Authenticated Link Model with Trusted Nodes

We first consider the case where nodes only rely on authenticated links and do not use digital signatures, i.e., $V_{auth}=\emptyset$ and $V_{n.auth}=V$. In this model, $V_{tc}=\emptyset$ because trusted components typically require digital signatures, but $V_t \subseteq V_{n.auth}$ is not necessarily empty (i.e., some nodes might be trusted).

We present our first protocol, `DolevU-T`, which extends `DolevU`, the state-of-the-art reliable communication protocol utilizing authenticated links, by leveraging trusted nodes. Notably, `DolevU` requires a $(2f+1)$-connected network, whereas `DolevU-T` also functions correctly on more generic networks in the presence of trusted nodes, as demonstrated by the example in App. C. We, therefore, characterize the networks on which `DolevU-T` is correct and provide two algorithms that take a network topology as input and indicate whether `DolevU-T` allows any pair of nodes to receive and authenticate each other's messages (i.e., whether it enforces RC).

3.1 Description of `DolevU-T`

Alg. 1 (in Appx. A) presents `DolevU-T`'s pseudocode. Messages contain two fields: the payload data m that is being broadcast and the dissemination path *path*. To broadcast, a node sends a given payload with an empty path to its neighbors (Alg. 1, l. 11), and delivers it. Upon receiving a message, a process p_i augments the received path variable with the sender's id (Alg. 1, l. 15). It then forwards the message to its neighbors that are not part of the received path (Alg. 1, l. 19). A process stores the paths it receives (Alg. 1, l. 17) and delivers a message if it was either received directly from the initial broadcaster or if it can identify $f+1$ vertex-disjoint paths among those it received (Alg. 1, l. 20). Receiving a payload through $f+1$ vertex-disjoint paths authenticates it, as there are at most f faulty nodes in the network. This implies that at least one of the $f+1$ disjoint paths through which the message traveled involved only correct nodes.

DolevU-T removes trusted nodes from a received path (Alg. 1, l. 16), before storing it in a *paths* set and attempting to identify $f+1$ vertex-disjoint paths. One should note that trusted nodes are not removed from the path that a node transmits to its neighbors (Alg. 1, l. 19) because it would make it possible for a node that receives a path to forward it to a neighbor that has already received it and has been removed from it, in turn leading to an unnecessary increase in the number of messages transmitted.

Removing trusted nodes from paths has interesting practical consequences. First, it reduces the size and the number of paths a node stores and manipulates. This is beneficial given that identifying disjoint paths has an exponential time complexity. Second, paths containing a shared list of trusted nodes might become disjoint. Finally, it might also generate an empty path from a path that contains only trusted nodes, which accelerates the delivery of a message through a protocol optimization presented in the next section.

3.2 Optimizations

The worst-case complexity of the original DolevU protocol is high, both in terms of message and computational complexity. Bonomi et al. presented five modifications to DolevU, referred to as MD.1–5, which reduce the number and size of transmitted messages in practical executions [9]:

MD.1 If a process p receives a message directly from the source s, then p directly delivers it.
MD.2 If a process p has delivered a message, then it discards all its related paths and relays the message with an empty path to all its neighbors.
MD.3 A process p relays a path related to a message only to the neighbors that have not delivered it (i.e., sent an empty path).
MD.4 If a process p receives a message with an empty path from a neighbor q, then p stops relaying and analyzing any path for that message that contains q.
MD.5 A process p stops relaying further paths related to a message after it has delivered it and has forwarded it with an empty path.

While DolevU-T can make use of MD.1–5, they are left out of Alg. 1 for simplicity. However, they do appear in the pseudocode of DualRC, our final protocol. DolevU-T can also leverage additional optimizations that prevent duplicate transmissions of a given message on a link (MBD.1 in [8]), and allow nodes to ignore messages that contain a *superpath* of a previously received path (MBD.10 in [8]), respectively.

3.3 Proofs

We start by proving that DolevU-T enforces *RC-no duplication* and *RC-no creation.*

▶ **Lemma 1.** *DolevU-T maintains **RC-no duplication** and **RC-no creation**.*

Proof. *RC-no duplication* is guaranteed by the use of the *delivered* variable. In DolevU, a node delivers a message when it receives it through $f+1$ vertex-disjoint paths, which guarantees that at least one of those paths only involves correct nodes. The presence of trusted nodes in a path does not affect whether contained nodes are correct. Thus, trusted nodes can be removed from a path used to attempt to authenticate a message, thereby proving *RC-no creation.* ◀

Note that `DolevU-T`, like `DolevU`, does not guarantee *RC-Validity* on all graphs. While it is known that `DolevU` requires a $(2f+1)$-connected communication graph, `DolevU-T`'s requirements are more complex.

Sec. 3.3.1 and 3.3.2 provide algorithms that determine whether `DolevU-T` provides *RC-Validity* on a given graph $G = (V, E)$, and therefore implements reliable communication on it. These algorithms rely on two functions $f(\cdot)$ and $g(\cdot)$ that we first define intuitively. Function $f(G)$, which is formally stated in Def. 2, takes a graph G as input and returns a new graph containing only the untrusted nodes of G. Furthermore, an edge is added between any two untrusted nodes if they are directly connected as neighbors in G or if they are connected by a path of trusted nodes in G. Function $g(G, G', u_t)$, defined as per Def. 3, inserts a trusted node $u_t \in V_t$ in a subgraph G' of a graph G and adds an edge in G' between u_t and a node of G' if they are neighbors or if they are connected by a path of trusted nodes in G.

▶ **Definition 2.** *Let f be the function that takes a graph $G = (V, E)$ as input and outputs the graph $f(G)=(V_0, E_0)$ such that:*

- $V_0 = V_{n.auth} \backslash V_t$
- $\forall (u, v) \in V_0^2,\ \big((u, v) \in E\big) \vee \Big(\exists u_{t_0}, \cdots, u_{t_n} \in V_t.(u, u_{t_0}) \in E \wedge (u_{t_n}, v) \in E$
  $\wedge \forall i < n.(u_{t_i}, u_{t_{i+1}}) \in E\Big) \Leftrightarrow (u, v) \in E_0$

In practice, the edges of a node $u \in V_0$ can be computed with a breadth-first traversal in G starting from u and following trusted edges. A faster approach determines E_0 by computing the connected components of a graph whose vertices correspond to those of G and whose edges are the edges from G that connect two trusted nodes. In such a graph, two untrusted nodes belong in the same connected component iff. they are connected in $f(G)$.

▶ **Definition 3.** *Let g be the function that takes as input a graph G, a subgraph $G'=(V' \subseteq V, E' \subseteq V^2)$ of G, and a trusted node $v_t \in V_t$, and outputs a graph $g(G, G', v_t)=(V_1, E_1)$ such that:*

- $V_1 = V' \cup \{v_t\}$
- $\forall (u, v) \in E',\ (u, v) \in E_1$ *(i.e., $E' \subseteq E_1$)*
- $\forall u \in V', \big((u, v_t) \in E\big) \vee \Big(\exists u_{t_0}, \cdots, u_{t_n} \in V_t,\ (u, u_{t_0}) \in E \wedge (u_{t_n}, v_t) \in E$
  $\wedge \forall i < n, (u_{t_i}, u_{t_{i+1}}) \in E\Big) \Leftrightarrow (u, v_t) \in E_1$

3.3.1 Method 1: Max-Flow on Transformed Graph

We now describe our first method, which verifies whether `DolevU-T` ensures reliable communication between any two nodes on a given network topology. Alg. 5 describes the corresponding pseudocode. This algorithm leverages the fact that the maximum number of edge-disjoint paths between any two nodes u and v is equal to the maximum flow between them when edges have unitary weights. To do so, it transforms graph G into a graph dG where a sufficient maximum flow between two nodes establishes their ability to communicate reliably using `DolevU-T` (i.e., its *RC-Validity*).

As we are interested in the number of vertex-disjoint paths between nodes, we must first modify the input graph G. Indeed, max-flow algorithms require edges to have a capacity and may use an edge up to its capacity, but several flows might cross a vertex. We, therefore, transform the graph using the node-splitting technique [19], which leads to the creation of a directed graph dG. More precisely, each node $u \in G$ is first split into two nodes u_{in} and u_{out}, which are connected by a directed edge $(u_{\text{in}}, u_{\text{out}})$ in dG. Then, an undirected edge (u, v) of G leads to the creation of two directed edges $(u_{\text{out}}, v_{\text{in}})$ and $(v_{\text{out}}, u_{\text{in}})$ in dG. In addition, all directed edges of dG are initially given a unitary capacity. We say that a node in dG is trusted if it has been created by splitting a trusted node of G. The capacity of all edges in dG that originate from a trusted node is set to $2f+1$.

▶ **Theorem 4.** *`DolevU-T` ensures RC-Validity in G iff. any two nodes $u, v \in G$ are either neighbors in G or if the flow in dG from u_{out} to v_{in} is larger than $2f+1$.*

Proof. As the flow from u_{out} to v_{in} in dG is equal to the flow from v_{out} to u_{in} by construction, we assume, w.l.o.g., that node u sends a message to node v.

($\Rightarrow$ Part 1/2) The first delivery condition is node v receiving an empty path. This implies that u and v are neighbors in G or connected through a non-empty path of trusted nodes in G. In the second case, the flow from u_{out} to v_{in} is at least $2f+1$ in dG because each edge from a trusted node has capacity $2f+1$.

($\Rightarrow$ Part 2/2) Node v can also deliver a message if it obtains $f+1$ vertex-disjoint paths after removing the trusted nodes from the paths it receives. In the presence of up to f faulty nodes, there should be $2f+1$ paths between u and v in G that become vertex-disjoint once their trusted nodes are removed. These $2f+1$ paths each have a capacity greater than or equal to 1. They might share edges that originate from a trusted node and have a capacity of $2f+1$, but not their other edges, which have a capacity of 1. Nonetheless, the existence of these $2f+1$ paths implies that the flow from u to v is at least $2f+1$ in dG.

($\Leftarrow$ Part 1/2) If u and v are neighbors in G then they can communicate reliably.

($\Leftarrow$ Part 2/2) If there is a flow larger than $2f+1$ between u_{out} to v_{in} in dG then u and v might first be connected by a path of trusted nodes in G. In that case, they can communicate reliably. Otherwise, there are $2f+1$ vertex-disjoint paths that may or may not contain trusted nodes connecting u_{out} to v_{in} in dG. These paths correspond to paths in G since an out node can only be reached by its corresponding in node in dG. In the presence of f faulty nodes, at least $f+1$ paths in G will therefore allow v to receive and authenticate a message sent by u. ◀

RC-Validity–Verification Algorithm (Alg. 5). To leverage Thm. 4, Alg. 5 determines whether all pairs (u, v) of nodes in a graph G can communicate reliably with `DolevU-T`. The algorithm first computes the directed graph dG from the input graph G. Each node u is split into two nodes $u_{\text{in}} = 2u$ and $u_{\text{out}} = 2u+1$ connected by a directed edge $(u_{\text{in}}, u_{\text{out}})$ of capacity $2f+1$ if u is trusted, or 1 otherwise (Alg. 5, ll. 8–13). The weighted edges of dG are then created from the edges of G: an edge $(u, v) \in E$ leads to two edges $(u_{\text{out}}, v_{\text{in}})$ and $(v_{\text{out}}, u_{\text{in}})$ in dG, and edges that originate from a trusted node are given capacity $2f+1$ and 1 otherwise (Alg. 5, ll. 15–19). Alg. 5 then studies all pairs of nodes $(u, v) \in V$ and checks if v can always authenticate a message sent by u. It first verifies whether u and v are neighbors in G or if a path of trusted nodes connects them in G (Alg. 5, ll. 23–28). This verification requires distinguishing between the various cases where u and/or v are trusted nodes, using functions f and g (cf. §6). Note that because the reliable communication abstraction assumes the broadcaster of a message to be correct, the weights of the outgoing edges of u need to be temporarily set to $2f+1$ (Alg. 5, l. 32) when u is considered the broadcaster (Alg. 5, l. 37). Alg. 5 then verifies whether the maximum flow between u and v is at least $2f+1$ in dG. If any check fails, then nodes (u, v) are not able to communicate reliably in the presence of f carefully chosen faulty nodes, and the algorithm returns FALSE (Alg. 5, l. 34). If the checks pass for all pairs of nodes in G, then Alg. 5 concludes that `DolevU-T` ensures *RC-Validity* and therefore implements reliable communication in graph G.

Complexity. App. B.2 computes the complexity of Alg. 5, which is $\mathcal{O}(f \cdot |V|^5)$.

3.3.2 Method 2: Eliminating Unnecessary Trusted Nodes from G and Checking for $2f+1$ Vertex-Connectivity

Our second correctness verification method improves upon the complexity of the previous approach when there are sufficient trusted nodes in the network. This second method relies on Thms. 5, 6 and 7 and also uses the aforementioned functions $f(\cdot)$ and $g(\cdot)$.

▶ **Theorem 5** (*RC-Validity*–untrusted ↔ untrusted)**.** *Two untrusted nodes $(u, v) \in (V_{nauth} \backslash V_t)^2$ can communicate reliably using `DolevU-T` in G iff. $u \in \Gamma_{f(G)}(v)$ or $\kappa_{f(G)}(u, v) \geq 2f+1$.*

Proof. *(⇒)* If u and v can communicate reliably using `DolevU-T` in G, then (i) they receive empty paths alongside each other's messages, implying they are neighbors in $f(G)$, or (ii) they are connected by $2f+1$ paths, which become vertex-disjoint once their trusted nodes are removed, and which connect them in $f(G)$.

(⇐ Part 1/2) Let us assume that u and v are neighbors in $f(G)$. By definition, either they are also neighbors in G, or they are connected by a path of trusted nodes in G. In both cases, they can communicate reliably in G using `DolevU-T` because they obtain empty paths alongside each other's messages and deliver them.

(⇐ Part 2/2) Let us now assume that u and v are connected through $2f+1$ vertex-disjoint paths in $f(G)$. Each of these paths is a subpath of at least a path that exists in G and may contain additional intermediary trusted nodes. `DolevU-T` removes trusted nodes from received paths, which means that u and v can always collect at least $f+1$ vertex-disjoint paths (that exist in $f(G)$) and authenticate each other's messages. ◀

▶ **Theorem 6** (*RC-Validity*–trusted ↔ untrusted)**.** *A trusted node $u_t \in V_t$ and an untrusted node $v \in V_{nauth} \backslash V_t$ can communicate reliably in G iff. $u_t \in \Gamma_{g(G,f(G),u_t)}(v)$ or $\kappa_{g(G,f(G),u_t)}(u_t, v) \geq 2f+1$.*

Proof. $g\big(G, f(G), u_t\big)$ inserts trusted node u_t in $f(G)$ and connects it with its untrusted neighbors in G and with the untrusted nodes it can communicate with through a path of trusted nodes in G. Substituting u by u_t, and $f(G)$ by $g\big(G, f(G), u_t\big)$ in the proof of Thm. 5 proves this theorem. ◀

▶ **Theorem 7** (*RC-Validity*–trusted ↔ trusted)**.** *Two trusted nodes $(u_t, v_t) \in V_t^2$ can communicate reliably in G iff. $u_t \in \Gamma_{g(G,g(G,f(G),u_t),v_t)}(v_t)$ or $\kappa_{g(G,g(G,f(G),u_t),v_t)}(u_t, v_t) \geq 2f+1$.*

Proof. Once again the proof is immediate after substituting u by u_t, v by v_t and $f(G)$ by $g\Big(G, g\big(G, f(G), u_t\big), v_t\Big)$ in the proof of Thm. 5. ◀

***RC-Validity*–Verification Algorithm (Alg. 6).** Alg. 6 directly leverages Thms. 5, 6, and 7 to verify whether `DolevU-T` ensures reliable communication between any pairs of nodes on a given graph G, assuming a set V_t of nodes are trusted and knowledge of the value f. The algorithm first verifies whether any two untrusted nodes can communicate reliably, which is the case if they are at least $2f+1$ vertex-connected or neighbors in $f(G)$ (Alg. 6, ll. 8–10). If so, Thm. 5 ensures that it will also be the case in G. Next, the algorithm similarly verifies whether each trusted node u_t can communicate reliably with all untrusted nodes, i.e., whether they are neighbors or $2f+1$ vertex-connected (Alg. 6, ll. 13–15) in $G_{u_t} = g(G, f(G), u_t)$. If so, Thm. 6 ensures that they can also communicate reliably in G. Finally, the algorithm verifies that any two trusted nodes u_t and v_t can communicate reliably in $G_{u_t,v_t} = g(G, G_{u_t}, v_t)$ (Alg. 6, ll. 16–19). Thm. 7 then ensures that u_t and v_t can communicate reliably in G. If False was not returned by the routine after executing all checks, the algorithm concludes that reliable communication is enforced by `DolevU-T` on graph G and returns True.

Complexity. App. B.3 computes the complexity of Alg. 6. When $|V_t|$ is small, the complexity is dominated by $|V \backslash V_t|^5 \approx |V|^5$. When $|V_t|$ is sufficiently large, the complexity is dominated by $|V_t|^2 \cdot |V| \approx |V|^3$, and Alg. 6 becomes faster than Alg. 5.

4 `SigFlood-T`: Reliable Communication in the Authenticated Process Model with Trusted Nodes

As a second stepping stone towards our final protocol, we now consider the system model where all nodes can use digital signatures (i.e., $V_{auth}{=}V$ and $V_{nauth}{=}\emptyset$). The state-of-the-art reliable communication protocol in unknown networks with authenticated processes uses network flooding to disseminate a payload signed by the original broadcaster. In this protocol, the sender of a broadcast signs its original message and sends it to all its neighbors. Upon receiving a payload and a valid signature, nodes verify its authenticity and forward both to all their neighbors that have not yet transmitted the message [26].

4.1 Description

We present the pseudocode of `SigFlood-T`, our reliable communication algorithm for the authenticated process model that leverages trusted nodes in unknown topologies, in Alg. 2 (Appx. A). Note that we assume all signatures to be correct for simplicity. In `SigFlood-T`, messages contain the payload data m that is being broadcast, the identity of the sender p_s and its signature $\sigma_s(m)$ (Alg. 2, l. 14). A node that receives a message from a trusted neighbor immediately delivers it, as it considers that its trusted neighbor authenticated it. Trusted components are not leveraged by `SigFlood-T`, since they do not have direct network access, which implies that their signature still needs verification.

In Alg. 2, the broadcasting node initially sends its message along with its signature to all its neighbors (Alg. 2, l. 9–13). When a node receives payload data along with a signature and the broadcaster's identity, it either verifies the correctness of the signature or accepts it immediately if the message is received directly from a trusted neighbor (Alg. 2, l. 16). Upon receiving and successfully verifying a message for the first time, a node forwards the message, along with its signature and the sender's id, to all its neighbors except the one from which the message was originally received.

Termination of `SigFlood-T` is guaranteed since each correct node forwards a given message at most once. `SigFlood-T` generates at most two messages per network link, which occurs when two neighbors simultaneously send a message to each other. In practice, it is therefore expected that `SigFlood-T` would generate significantly fewer messages than `DolevU-T` on a given network topology, whose base version generates $O(|V|!)$ messages.

4.2 Proofs

Lemmas 8 and 9 determine when `SigFlood-T` implements reliable communication.

▶ **Lemma 8.** *`SigFlood-T` ensures **RC-no duplication** and **RC-no creation**.*

Proof. *RC-no duplication* is guaranteed by the use of the *delivered* variable. *RC-no creation* is ensured because a message is delivered only when the signature of its creator is verified. ◀

▶ **Lemma 9.** *Network topologies on which `SigFlood-T` ensures **RC-Validity** can be identified using either Alg. 6 or Alg. 5.*

Proof. In the absence of trusted nodes, `SigFlood-T` requires the network to be at least $(f+1)$ connected to ensure that the broadcaster's signature reaches every node. Conversely, `DolevU-T` requires the network to be at least $(2f+1)$ connected. Verifying the correctness of `SigFlood-T` on a graph can be done using Alg. 6 or Alg. 5 by replacing all instances of $2f+1$ with $f+1$ in those algorithms. ◀

5 `DualRC`: Reliable Communication in the Hybrid Authenticated Links/Processes Model with trusted nodes and components

We now consider the most generic scenario that involves both non-authenticated and authenticated nodes. Our final algorithm, `DualRC`, builds upon our two previous protocols, `DolevU-T` and `SigFlood-T`, and therefore inherits their ability to leverage trusted nodes. However, contrary to these two protocols, `DualRC` also uses specific optimizations that require manipulation of both dissemination paths and signatures. In particular, `DualRC` leverages the presence of trusted components by allowing them to transform a set of signed and disjoint dissemination paths into a single trusted signature. Alg. 3 (Appx. A) lays out the pseudocode of `DualRC`, where the blue text is specific to the presence of trusted components. Once again, we omit the handling of incorrect signatures for simplicity.

5.1 Description

Message types. `DualRC` uses both dissemination paths and digital signatures to allow all nodes to authenticate a message. `DualRC` manipulates the following two types of messages:

- A *signature message* $[m, s, \sigma_l(m, s)]$ contains a payload m, the id s of the original sender p_s and a signature generated by process p_l over (m, s). All processes can manipulate and interpret signature messages in order to deliver messages.
- A *path message* $[m, s, path, List(pa, \sigma(pa, m, s))]$ contains a payload m, the id s of the original sender p_s, an unsigned dissemination path *path*, and a possibly empty list of signed subpaths $List(pa, \sigma(pa, m, s))$. Path messages can only be interpreted by authenticated processes, but non-authenticated processes might have to forward them. Modifications MD.1-MD.5 (cf. §3.2) are used when manipulating path messages. As an optimization, a list of signed paths could be compressed into a single path with an onion signature.

Initialization. Each process maintains several variables. The *delivered* boolean indicates whether the message has been delivered (init. False); the sets *paths* and *signedPaths* contain, respectively, the received unsigned and signed dissemination paths (init. $\emptyset$); dN is the set of nodes that are known to have delivered the message (init. $\emptyset$); *sigs* is a set that contains the signatures that have been received (init. $\emptyset$).

Broadcasting a message. When a node p_i broadcasts a payload m, it sends a path message $[m, i, \emptyset_p, \emptyset_l]$ to all its neighbors (Alg. 3, l. 25). This message carries an empty dissemination path $\emptyset_p$ and an empty list $\emptyset_l$ of signed subpaths. If node p_i is authenticated then it additionally sends a signature message $[m, i, \sigma_i(m, i)]$ to its neighbors (Alg. 3, l. 24). Node p_i then immediately delivers m. Upon reception, signature and path messages are handled by distinct procedures, executed by all types of nodes.

Receiving a signature message $[m, s, \sigma_l(m, s)]$ (Alg. 3, l. 32). A node signs a message only after it has been delivered. Received signatures are stored in the locally maintained set *sigs* and processed exactly once. A message is delivered without signature verification if it has been either received directly from the broadcaster or relayed by a trusted and authenticated node (Alg. 3, l. 34). Authenticated nodes can also deliver m using signature $\sigma_l(m, s)$ if it was generated by the broadcaster ($p_s = p_l$), by a trusted node or component, or if they identify $f+1$ disjoint signed paths in *sPaths* (a signature is interpreted as a single-node signed path, except for the broadcaster's signature, which is interpreted as an empty path) (Alg. 3, l. 42). If so, a node then sends a signature to the subset $dests = \Gamma_G(p_i) \setminus \{p_s, p_l\}$ of its neighbors, which might not have delivered m. If $\sigma_l(m, s)$ is a trusted signature then p_i forwards it to the nodes in *dests* and returns (Alg. 3, l. 46). Otherwise, if the process is equipped with a trusted component, it verifies $\sigma_l(m, s)$ and generates a trusted signature $\sigma_{TC_i}(m, s)$ that it then forwarded to the nodes in *dests* and returns (Alg. 3, l. 50). If it is authenticated, and if it has not forwarded or generated a trusted node's or component's signature, then node p_i sends its own signature to the nodes in *dests* (Alg. 3, l. 55). Finally, a node that has not returned forwards the signature it received to the processes in *dests* (Alg. 3, l. 58).

Receiving a path message $[\boldsymbol{m, s, path, List(pa, \sigma(pa, m, s))}]$ (Alg. 3, l. 60). A node might ignore a path message using MD.4 or MD.5 as in `DolevU-T`. If *path* is empty, then node p_j is added to dN, which tracks the nodes that delivered a path message.

If node p_i is authenticated, then it tries to leverage each path *pa* and signed path $\sigma_l(pa, m, s)$ contained in $List(pa, \sigma(pa, m, s))$ (Alg. 3, ll. 65–76). First, the path $pa \cup \{p\}$ – of which all trusted signatures are removed – is added to *sPaths*. If *sPaths* contains an empty path (i.e., the broadcaster's signature was received) or $f+1$ vertex-disjoint paths, then node p_i delivers m. Node p_i then broadcasts its trusted component's signature, if it is equipped with one, and otherwise broadcasts its own signature in a signature message. More precisely, node p_i transfers the $f+1$ signed vertex-disjoint paths it identified to its trusted component so that it can verify their signatures, check that they are disjoint, and if so return a signature $\sigma_{TC_i}(m, s)$ (Alg. 3, ll. 50 and 71). Node i then also broadcasts a path message with an empty path to all its neighbors in $\Gamma_G(p_i) \setminus dN$ and returns.

If node p_i has not delivered m using signed paths, it attempts to do so with dissemination path *path* (Alg. 3, ll. 78–92). This path is obtained from the received path to which p_j is added and from which all trusted nodes are removed (Alg. 3, l. 79). Note that processes equipped with a trusted component cannot be removed from unsigned dissemination paths because they do not manipulate them. If node p_i delivers m based on a dissemination path, then it broadcasts its signature to all its neighbors in a signature message and broadcasts a path message that contains an empty path and the list of signed paths it received to which an appended signed path has been added to the nodes in $\Gamma_G(p_i) \setminus dN$. If node p_i is not authenticated then it only broadcasts a path message with an empty path and the received list of signed paths. A node that delivers using *path* returns.

Finally, if node p_i has not delivered payload m, it forwards the message it received with up to two modifications: it sets path $rpath = path \cup \{p_j\}$, and if it is authenticated, it appends $\sigma_i(rpath, m, s)$ to the list of signed paths (Alg. 3, ll. 94–98).

Termination and Optimizations. A node stops processing and forwarding path messages as soon as it delivers (as in `DolevU-T`). However, a node still has to forward signatures after delivering to allow all authenticated nodes to deliver (App. D shows an example where this is necessary). Termination is ensured because a node processes and relays a signature at most once through the use of variable *sigs*.

Limiting further dissemination of signatures, particularly by non-authenticated nodes, is more challenging, and the corresponding optimizations were omitted from Alg. 3 due to space constraints. First, authenticated nodes can stop forwarding signature messages after relaying the broadcaster's signature, a trusted signature, or $f+1$ untrusted signatures, as transmitting additional signatures provides no further information (the message is authenticated). Conversely, a signature should not be forwarded to a neighbor that sent the broadcaster's signature, a trusted signature, or $f+1$ untrusted signatures. Non-authenticated nodes can stop forwarding signatures after they have relayed $f+1$ different signatures that they can reliably trace back to $f+1$ different authenticated nodes that relayed them or emitted them. Indeed, each of these untrusted signatures has been verified by $f+1$ nodes, and at least one can be correctly assumed to have successfully and correctly verified it.

Another optimization involves encoding the list of signed paths within a message as a single onion signature or an aggregate signature. This would reduce the bit complexity of `DualRC` effectively. As a dissemination path should not be transferred to a neighbor that delivered (MD.3), any node (i.e., authenticated or not) can avoid sending path messages to a node that transmitted the broadcaster's signature, a trusted signature, its own signature, or $f+1$ untrusted signatures. Finally, it might be possible to avoid creating unsigned path messages between authenticated nodes, but we believe that it would render `DualRC`'s code significantly more complex.

5.2 Proofs

Delivery requirements. `DualRC`'s pseudocode (Alg. 3, Appx. A) allows a node to deliver a message under two conditions (DR.1 & DR.2):

[DR.1]: A non-authenticated node v delivers a message from a broadcaster u if at least one of subconditions NA.1 and NA.2 is met.

NA.1 Node v computes an empty path after removing the trusted nodes it may contain.

NA.2 Node v computes $f+1$ vertex-disjoint paths after removing all trusted nodes from them.

[DR.2] An authenticated node v delivers a message from a broadcaster u if subconditions NA.1 or NA.2 are met or if at least one of subconditions A.1 or A.2 is met.

A.1 Node v receives node u's, a trusted node's, or a trusted component's signature.

A.2 Node v computes a combination of $f+1$ signatures or vertex-disjoint paths, after removing all trusted nodes from them.

Note that A.2 generalizes NA.2 for authenticated nodes. Unfortunately, conditions NA.1, NA.2. A.1, and A.2 do not directly lead to a correctness verification algorithm one could apply on a given graph. To reach this objective, we formulate equivalent delivery conditions in Thm. 10. These conditions assume a set S of nodes known to have delivered a message broadcast by a process u and require manipulation of the directed graph dG, generated as described in §3.3.1, so that max-flows can be computed. As a brief summary, graph dG is obtained from g using the following steps: (i) nodes are split into two, and edges are added between them with a capacity of $2f+1$ if the original node is trusted or 1 otherwise, and (ii) edges of the original graph G are transformed into two directed edges in dG of weight $2f+1$ if their origin is trusted or 1 otherwise. We introduce synthetic *root* nodes that are connected to different subsets of nodes in S in graph dG using weighted directed edges (from a root to the nodes):

- Node $root_S$ is connected to all nodes in $S \cap V$ using edges of capacity 1 (for untrusted nodes) or $2f+1$ for trusted nodes.
- Node $root_{n.auth}$ is connected to all (non-authenticated) nodes in $S \cap V_{n.auth}$ with edges of weight 1.

▶ **Theorem 10.** *A node v is guaranteed to always deliver a message that is broadcast by a (correct) node u iff. at least one of the following conditions is verified.*

5.1 *Nodes u and v are neighbors in G.*
5.2 *The max flow from $root_S$ to v in dG is at least $2f{+}1$.*
5.3 *Nodes u and v are authenticated, and the max flow from u to v in dG is larger than $f{+}1$.*
5.4 *Node v is authenticated, and there is at least one trusted and authenticated node in S whose max flow in dG to v is greater than or equal to $f{+}1$.*

Proof. *(⇒)* Condition NA.1 can be met in two cases: (i) u and v are neighbors (condition 5.1), or (ii) they are connected by a path of trusted nodes (a subcase of condition 5.2).

Condition NA.2 can occur following several scenarios. First, if there are $2f{+}1$ vertex-disjoint paths from u to v. Second, if at least $2f{+}1$ untrusted nodes deliver a message and, therefore, send an empty path to their neighbors as a result of which eventually v always receives $f{+}1$ disjoint paths. Third, if v is able to receive at least $2f{+}1$ non-empty dissemination paths that originate from trusted nodes. Finally, if v should be able to receive a combination of $2f{+}1$ vertex-disjoint paths and signatures, that are also used as single-node paths. In each case, the flow from $root_S$ to v in dG is at least $2f + 1$ (condition 5.2), which implies that $f{+}1$ nodes in S (possibly including u) are correct and will broadcast empty paths. Node v will then always receive $f{+}1$ disjoint paths as a consequence of these delivering nodes. As dissemination paths are not relayed by nodes that deliver a message, one could wonder whether v will always receive these $f{+}1$ disjoint paths. This is indeed the case because a delivering node located on one of the $f{+}1$ correct paths would still broadcast an empty path that would reach v.

The first subcase of condition A.1 is condition 5.3, while the second one is condition 5.4. Note that the presence of trusted components in the network does not modify `DualRC`'s connectivity requirement, as the host of a trusted component can still avoid sending messages.

(⇐) Two nodes that are neighbors can always reliably communicate (condition 5.1). If condition 5.2 is verified, then v will receive $f + 1$ disjoint dissemination paths. If condition 5.3 is verified, then node v will receive the signature of node u. Finally, if condition 5.4 is verified, then node v will receive a trusted signature. ◀

***RC-Validity*–Verification Algorithm.** Our correctness verification algorithm relies on conditions 5.1–5.4 to maintain the set of nodes S that are guaranteed to be able to deliver a broadcaster's message in the presence of up to f faulty nodes. Initially, S only contains the broadcaster. The algorithm then successively attempts to extend S by identifying nodes that will satisfy one of the conditions NA.1, NA.2, A.1, or A.2 until all nodes have been added to it (in which case reliable communication is proven), or until it cannot grow anymore (in which case there exists a set of f nodes that, if faulty, would prevent some nodes from authenticating each other's messages). Checking `DualRC`'s correctness on a given graph requires executing our algorithm for each possible broadcasting node.

Complexity. The complexity of this verification algorithm is $\mathcal{O}(|V|^5)$.

6 Related Work

Reliable communication (RC) in the global fault model. RC was first studied assuming authenticated links in partially connected and unknown networks. Dolev showed that correct processes can communicate reliably in the presence of f Byzantine nodes iff. the network graph is $(2f{+}1)$-vertex connected [16]. Their algorithm that assumes an unknown network topology, DolevU, propagates a message along all possible dissemination paths. A variant of this algorithm leverages a routed dissemination in known networks. Beimel et al. [5, 6] considered the routed version of Dolev's algorithm, extending it to scenarios where node pairs can authenticate each other's messages. We consider that the topology is unknown, where any authenticated node can authenticate any signed message. More recently, Bonomi et al. improved the expected performance of DolevU using several optimizations [9]. In the authenticated process model, processes can generate a signature for each message they send and rely on network flooding to reach all possible destinations. In this work, we consider a static network with the global fault model and leverage both the DolevU and network flooding-based RC algorithms.

Reliable communication in the local fault model. Koo presented a broadcast algorithm under the t-locally bounded fault model [27], which was later coined the Certified Propagation Algorithm (CPA) [35]. Maurer and Tixeuil have also defined weaker reliable communication primitives that benefit from higher scalability [29, 30]. Recently, Bonomi et al. [11] established conditions for CPA's correctness in dynamic communication networks.

Network requirements for reliable communication. In the authenticated link model, DolevU requires a $2f+1$ connected [16] communication network. This network connectivity requirement in the authenticated process model is lowered to $f+1$. Several works identified and refined graph-theoretic parameters for CPA to be correct in the local fault model [23, 28, 35, 38]. In this work, we observe that the presence of trusted processes, along with both authenticated and non-authenticated processes, weakens the network connectivity requirements for reliable communication in the global fault model. Additionally, we identify sufficient conditions for this to occur.

Hybrid Byzantine-crash fault model. Hadzilacos proposed a model in which processes may fail by either halting or omitting some messages [22]. Thambidurai and Park [36], and Meyer and Pradhan [33] considered the agreement problem under a hybrid fault model that involves crashing and Byzantine processes. Garay and Perry studied reliable broadcast and consensus when up to f nodes are either Byzantine or fail by crashing [21]. Backes and Cachin described a reliable broadcast protocol that considers a hybrid model where up to f Byzantine nodes and t nodes can crash and recover [4].

Hybrid Byzantine-trusted fault model. More recently, Tseng et al. modified CPA to leverage trusted nodes [39, 40]. Abraham et al. described an authenticated multivalued consensus protocol in synchronous networks resilient to a mix of Byzantine and crash faults [2]. Differently, the hybridization approach equips nodes that run an algorithm with trusted components to improve its resilience [41]. Several consensus algorithms consider hybrid models that involve authenticated nodes that leverage a trusted component [14, 15, 25, 42]. Nowadays, trusted components have been implemented with technologies such as Intel SGX [24], ARM TrustZone [3], or TPMs [37]. As far as we know, our work is the first to leverage a trusted component for reliable communication.

7 Conclusion

We proposed novel reliable communication protocols for incomplete network topologies in the global fault model. We first extended the state-of-the-art protocols that have respectively been designed for the authenticated process and authenticated link models so that they can tolerate and leverage the possible presence of trusted processes. We then showed that these protocols can be non-trivially combined into `DualRC`, our novel reliable communication protocol that allows both authenticated and non-authenticated processes to communicate. `DualRC` leverages the signatures of authenticated processes and the presence of trusted components or trusted processes. `DualRC` provides reliable communication over a larger set of network topologies than previous protocols. Last, we also described methods that one can use to verify whether reliable communication is possible with our algorithms on a given network topology and evaluated the complexity of these methods.

References

1 Waseem Abbas, Aron Laszka, and Xenofon Koutsoukos. Improving network connectivity and robustness using trusted nodes with application to resilient consensus. *IEEE Transactions on Control of Network Systems*, 5(4):2036–2048, 2017. doi:10.1109/TCNS.2017.2782486.

2 Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. Authenticated consensus in synchronous systems with mixed faults. *Cryptology ePrint Archive*, 2022.

3 ARM. Arm security technology: building a secure system using trustzone technology, 2009.

4 Michael Backes and Christian Cachin. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *DSN*, 2003.

5 Amos Beimel and Matthew Franklin. Reliable communication over partially authenticated networks. *Theoretical computer science*, 220(1):185–210, 1999. doi:10.1016/S0304-3975(98)00241-2.

6 Amos Beimel and Lior Malka. Efficient reliable communication over partially authenticated networks. *Distributed Computing*, 18(1):1, 2005.

7 Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015. doi:10.1137/14097032X.

8 Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. Practical byzantine reliable broadcast on partially connected networks. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 506–516. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00055.

9 Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Multi-hop byzantine reliable broadcast with honest dealer made practical. *Journal of the Brazilian Computer Society*, 25(1):9:1–9:23, 2019. doi:10.1186/S13173-019-0090-X.

10 Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Boosting the efficiency of byzantine-tolerant reliable communication. In *SSS*, 2020.

11 Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Reliable communication in dynamic networks with locally bounded byzantine faults. *Journal of Parallel and Distributed Computing*, 193:104952, 2024.

12 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

13 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

14 Miguel Correia, Lau Cheuk Lung, Nuno Neves, and Paulo Veríssimo. Efficient byzantine-resilient reliable multicast on a hybrid failure model. In *SRDS*, 2002.

15 Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. Damysus: streamlined BFT consensus leveraging trusted components. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *European Conference on Computer Systems*, pages 1–16. ACM, 2022. doi:10.1145/3492321.3519568.

16 Danny Dolev. Unanimity in an unknown and unreliable environment. In *FOCS*. IEEE, 1981.

17 Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004. doi:10.1109/MC.2004.1297243.

18 Lester Randolph Ford and Delbert Ray Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

19 Lester Randolph Ford and Delbert Ray Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N.J., 1962.

20 Bernard A. Galler and Michael J. Fischer. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, 1964. doi:10.1145/364099.364331.

21 Juan A Garay and Kenneth J Perry. A continuum of failure models for distributed computing. In *International Workshop on Distributed Algorithms*, pages 153–165. Springer, 1992. doi:10.1007/3-540-56188-9_11.

22 Vassos Hadzilacos. Issues of fault tolerance in concurrent computations. *Dissertation Abstracts International*, 46(7), 1986.

23 Akira Ichimura and Maiko Shigeno. A new parameter for a broadcast algorithm with locally bounded byzantine faults. *Information processing letters*, 110(12-13):514–517, 2010. URL: https://core.ac.uk/download/pdf/56648228.pdf, doi:10.1016/J.IPL.2010.04.003.

24 Intel SGX. URL: https://software.intel.com/en-us/sgx.

25 Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: Resource-efficient byzantine fault tolerance. In *EuroSys*, 2012.

26 Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *ACM SIGOPS Oper. Syst. Rev.*, 41(5):2–7, 2007. doi:10.1145/1317379.1317381.

27 Chiu-Yuen Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *PODC*, pages 275–282, 2004. doi:10.1145/1011767.1011807.

28 Chris Litsas, Aris Pagourtzis, and Dimitris Sakavalas. A graph parameter that matches the resilience of the certified propagation algorithm. In *AdHoc-Now*, 2013.

29 Alexandre Maurer and Sébastien Tixeuil. Byzantine broadcast with fixed disjoint paths. *Journal of Parallel and Distributed Computing*, 74(11):3153–3160, 2014. doi:10.1016/J.JPDC.2014.07.010.

30 Alexandre Maurer and Sébastien Tixeuil. Containing byzantine failures with control zones. *IEEE TPDS*, 26(2):362–370, 2014. doi:10.1109/TPDS.2014.2308190.

31 Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. Attestation mechanisms for trusted execution environments demystified. In *DisCoTec*, 2022.

32 Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

33 Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE TPDS*, 2(02):214–222, 1991. doi:10.1109/71.89066.

34 Tal Navon and David Peleg. Mixed fault tolerance in server assignment: Combining reinforcement and backup. *Theoretical Computer Science*, 836:76–93, 2020. doi:10.1016/J.TCS.2020.06.033.

35 Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, 2005. doi:10.1016/J.IPL.2004.10.007.

36 Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *SRDS*. IEEE, 1988.

37 Trusted platform module, 2016. URL: https://trustedcomputinggroup.org/resource/tpm-library-specification/.

38 Lewis Tseng, Nitin Vaidya, and Vartika Bhandari. Broadcast using certified propagation algorithm in presence of byzantine faults. *Information Processing Letters*, 115(4):512–514, 2015. doi:10.1016/J.IPL.2014.11.010.

39 Lewis Tseng, Yingjian Wu, Haochen Pan, Moayad Aloqaily, and Azzedine Boukerche. Reliable broadcast in networks with trusted nodes. In *GLOBECOM*. IEEE, 2019.

40 Lewis Tseng, Yingjian Wu, Haochen Pan, Moayad Aloqaily, and Azzedine Boukerche. Reliable broadcast with trusted nodes: Energy reduction, resilience, and speed. *Computer Networks*, 182:107486, 2020. doi:10.1016/J.COMNET.2020.107486.

41 Paulo E Veríssimo. Travelling through wormholes: a new look at distributed systems models. *ACM SIGACT News*, 37(1):66–81, 2006. doi:10.1145/1122480.1122497.

42 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.

43 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Symposium on Principles of Distributed Computing*. ACM, 2019. doi:10.1145/3293611.3331591.

A Pseudocodes of DolevU-T, SigFlood-T and DualRC

In this section, we provide the pseudocodes for our reliable communication algorithms. For both the authenticated channel model (DolevU-T) and the authenticated process model (SigFlood-T) that leverage trusted nodes. Alg. 1 presents DolevU-T's pseudocode, which is discussed in Sec. 3. Alg. 2 presents SigFlood-T's pseudocode, which we discuss in Sec. 4. Alg. 3 presents DualRC's pseudocode, which is discussed in Sec. 5.

Algorithm 1 DolevU-T: Reliable communication in $(2f+1)$-connected networks at process p_i with trusted processes in the authenticated link model.

```
1:  Parameters:
2:      $f$ : max. number of Byzantine processes in the system.
3:  Uses:
4:      • Auth. async. perfect point-to-point links, instance $al$.
5:      • $\Gamma_G(p_j)$ returns the list of $p_j$'s neighbors.

6:  upon event ⟨DolevU-T, Init⟩ do
7:      $delivered = $ False  ◁ to deliver at most once
8:      $paths = \emptyset$  ◁ received dissemination paths

9:  upon event ⟨DolevU-T, Broadcast | $m$⟩ do
10:     forall $p_j \in \Gamma_G(p_i)$ do
11:         $\langle al,$ Send $| p_j, [m, \emptyset_p]\rangle$
12:     ⟨DolevU-T, Deliver | $m$⟩
13:     $delivered = $ True

14: upon event $\langle al,$ Receive $| p_j, [m, path]\rangle$ do
15:     $fwd\_path = path + [p_j]$
16:     $path = $ remove_all_trusted($fwd\_path$)
17:     $paths$.insert($path \setminus \{p_j\}$)
18:     forall $p_k \in \Gamma_G(p_i) \setminus path$ do
19:         $\langle al,$ Send $| p_k, [m, fwd\_path]\rangle$

20: upon event ($paths$ contains [ ] (empty path) or $f+1$ vertex-disjoint paths)
    and (not $delivered$) do
21:     ⟨DolevU-T, Deliver | $m$⟩
22:     $delivered = $ True
```

Algorithm 2 `SigFlood-T`: Reliable communication in $(f+1)$-connected networks at process p_i in the presence of trusted processes in the authenticated process model.

```
 1: Parameters:
 2:     $f$ : max. number of Byzantine processes in the system.
 3: Uses:
 4:     • Auth. async. perfect point-to-point links, instance $al$.
 5:     • $G = (V, E)$: network topology
 6:     • $\sigma_i(m)$ is the signature of process $p_i$ over message $m$, and verif($\sigma_j(m)$) verifies it.

 7: upon event ⟨SigFlood-T, Init⟩ do
 8:     $delivered$ = FALSE

 9: upon event ⟨SigFlood-T, Broadcast | $m$⟩ do
10:     forall $p_j \in \Gamma_G(p_i)$ do
11:         ⟨$al$, Send | $p_j$, [$m, \sigma_i(m)$]⟩
12:     $delivered$ = TRUE
13:     ⟨SigFlood-T, Deliver | $m$⟩

14: upon event ⟨$al$, Receive | $p_j$, [$m, \sigma_s(m)$]⟩ do
15:     if $delivered$ then return
16:     if isTrusted($p_j$) or verif($\sigma_s(m)$) then
17:         $delivered$ = TRUE
18:         ⟨SigFlood-T, Deliver | $m$⟩
19:         forall $p_k \in \Gamma_G(p_i) \setminus \{p_j, p_s\}$ do
20:             ⟨$al$, Send | $p_k$, [$m, \sigma_s(m)$]⟩
```

Algorithm 3 `DualRC`: Reliable communication at process p_i with trusted and TC-augmented processes in the hybrid authenticated link and process model. Modifications MD.1–5 are used with dissemination paths (cf. §3.2). Blue text is specific to the presence of trusted components.

```
Parameters:
    $f$ : max. number of Byzantine nodes in the system.
Uses:
    • Auth. async. perfect point-to-point links, instance $al$.
    • $G = (V, E)$: network topology
    • $\Gamma_G(p_j)$ returns the list of $p_j$'s neighbors.
    • isTrusted($p_j$) indicates if $p_j$ is trusted.
    • hasTC($p_j$) indicates if $p_j$ has a trusted component.
    • isAuth($p_j$) indicates if $p_j$ is authenticated.
    • isTC($\sigma$) indicates if a signature was generated by a TC.
    • (Auth. nodes only) $\sigma_i(m)$ is the signature of process $p_i$
over message $m$, and verif($\sigma_j(m)$) verifies it.
    • (Nodes equipped with a TC) $\sigma_{TC_i}(m)$ is the signature
of $p_i$'s TC.

upon event⟨DualRC, Init⟩ do
$delivered$ = FALSE
$paths = \emptyset$  ◁ Received dissemination paths
$sPaths = \emptyset$  ◁ Received signed dissemination paths
$dN = \emptyset$  ◁ Set of nodes that delivered (for MD.3)
$sigs = \emptyset$  ◁ Set of received signatures

upon event ⟨DualRC, Broadcast | $m$⟩ do
forall $p_j \in \Gamma_G(p_i)$ do
    if isAuth($p_i$) then
        ⟨$al$, Send | $p_j$, $[m, i, \sigma_i(m, i)]$⟩
    ⟨$al$, Send | $p_j$, $[m, i, \emptyset_p, \emptyset_l]$⟩
deliver($m$)

function deliver($m$)
if not $delivered$ then
    $delivered$ = TRUE
    ⟨DualRC, Deliver | $m$⟩

 ◁ Receive a single signature
upon event ⟨$al$, Receive | $p_j$, $[m, s, \sigma_l(m, s)]$⟩ do
if $\sigma_l(m, s) \in sigs$ then return; else $sigs$=$sigs \cup \{\sigma_l(m, s)\}$
if $p_j == p_l$ or (isTrusted($p_j$) and isAuth($p_j$)) then
    $dN = dN \cup \{p_l\}$
    deliver($m$)

$dests = \Gamma_G(p_i) \setminus \{p_s, p_l\}$
if $isAuth(p_i)$ then  ◁ try to deliver and broadcast own sig.
    $sPaths$.insert($\{p_l\} \setminus \{p_s\}$)
    $sPaths\_cond$=$\exists\{f$+1 vertex-disjoint paths$\}\subset sPaths$
    if ($p_s == p_l$ or isTrusted($p_l$) or
    isTC($\sigma_l(m)$) or $sPaths\_cond$) then
        deliver($m$)
        if isTrusted($p_l$) or isTC($\sigma_l(m, s)$) then
            forall $p_k \in dests$ do
                ⟨$al$, Send | $p_k$, $[m, s, \sigma_l(m, s)]$⟩  ◁ fwd sender's sig.
            return
        else if hasTC($p_i$) then
            forall $p_k \in dests$ do
                ⟨$al$, Send | $p_k$, $[m, s, \sigma_{TC_i}(m, s)]$⟩  ◁ send TC's sig.
            $sigs = sigs \cup \{\sigma_{TC_i}(m, s)\}$
            return
        else
            forall $p_k \in dests$ do
                ⟨$al$, Send | $p_k$, $[m, s, \sigma_i(m, s)]$⟩  ◁ send own sig.
            $sigs = sigs \cup \{\sigma_i(m, s)\}$
```

Algorithm 4 `DualRC` (continued).

```
57: forall $p_k \in dests$ do
58:     $\langle al, \texttt{Send} \mid p_k, [m, s, \sigma_l(m, s)]\rangle$ $\triangleleft$ forward (non-trusted) sig.

59: $\triangleleft$ Receive a single path, with signatures on subpaths
60: upon event $\langle al, \texttt{Receive} \mid p_j, [m, s, path, List(pa, \sigma(pa, m, s))]\rangle$ do
61: if $delivered$ or $path \cap dN \neq \emptyset$ then return $\triangleleft$ MD.5 and MD.4
62: if $path == \emptyset_p$ then $dN = dN \cup \{p_j\}$ $\triangleleft$ Check if dN is correct - MD.3

63: $\triangleleft$ Try to identify $f$+1 signed vertex-disjoint paths.
64: if $\texttt{isAuth}(p_i)$ then
65:     forall $pa, \sigma_l(pa, m, s) \in List(pa, \sigma(pa, m, s))$ do
66:         $sPaths.\texttt{insert}(\texttt{rm\_all\_trusted\_TCs}(pa \cup \{p_l\}))$
67:         if $\emptyset_p \in sPaths$ or $\exists\{f{+}1 \text{ vertex-disjoint paths}\} \subset sPaths$ then
68:             $\texttt{deliver}(m)$
69:             forall $p_k \in \Gamma_G(p_i)$ do
70:                 if $\texttt{hasTC}(p_i)$ then
71:                     $\langle al, \texttt{Send} \mid p_k, [m, s, \sigma_{TC_i}(m, s)]\rangle$
72:                 else
73:                     $\langle al, \texttt{Send} \mid p_k, [m, s, \sigma_i(m, s)]\rangle$
74:             forall $p_k \in \Gamma_G(p_i) \backslash dN$ do $\triangleleft$ MD.2 + MD.3
75:                 $\langle al, \texttt{Send} | p_k, [m, s, \emptyset_p, \sigma_i(\emptyset_p, m, s) + List(pa, \sigma(pa, m, s))]\rangle$
76:             return

77: $\triangleleft$ $p_i$ is not authenticated or signed paths did not cause a deliver
78: $rpath = path \cup \{p_j\}$ $\triangleleft$ path to forward
79: $path = \texttt{remove\_all\_trusted}(rpath)$
80: $paths.\texttt{insert}(path)$

81: $\triangleleft$ try to deliver using dissemination paths
82: if $\emptyset_p \in paths$ (MD.1) or $\exists\{f{+}1 \text{ vertex-disjoint paths}\} \subset paths$ then
83:     if $\texttt{isAuth}(p_i)$ then
84:         forall $p_k \in \Gamma_G(p_i)$ do $\triangleleft$ MD.2-3
85:             $\langle al, \texttt{Send} \mid p_k, [m, s, \sigma_i(m, s)]\rangle$
86:         forall $p_k \in \Gamma_G(p_i) \backslash dN$ do $\triangleleft$ MD.2-3
87:             $\langle al, \texttt{Send} \mid p_k, [m, s, \emptyset_p, \sigma_i(\emptyset_p, m, s) + List(pa, \sigma(pa, m, s))]\rangle$
88:     else
89:         forall $p_k \in \Gamma_G(p_i) \backslash dN$ do $\triangleleft$ MD.2-3
90:             $\langle al, \texttt{Send} \mid p_k, [m, s, \emptyset_p, List(pa, \sigma(pa, m, s))]\rangle$
91:     $\texttt{deliver}(m)$
92:     return

93: $\triangleleft$ If not delivered, forward message with modified path
94: forall $p_l \in \Gamma_G(p_i) \setminus rpath$ do
95:     if $\texttt{isAuth}(p_i)$ then
96:         $\langle al, \texttt{Send} \mid p_l, [m, s, rpath, \sigma_i(rpath, m, s) + List(pa, \sigma(pa, m, s))]\rangle$
97:     else
98:         $\langle al, \texttt{Send} \mid p_l, [m, s, rpath, List(pa, \sigma(pa, m, s))]\rangle$
```

B Correctness Verification of `DolevU-T`

In this section, we provide the pseudocodes for our two algorithms that verify whether `DolevU-T` correctly implements the validity property of reliable communication on a given graph. These two algorithms are discussed and proven in Sec. 3.3.1 and 3.3.2, respectively. We then evaluate the complexity of our two algorithms.

B.1 Pseudocodes

Algorithm 5 Max-flow-based verification of `DolevU-T`'s *RC-Validity* on a graph (`DolevU-T-Verif`, method 1).

```
Inputs:
    $G = (V, E)$: undirected network topology.
    $V_t \subset V$: set of trusted nodes in $G$.
    $f$: max. number of Byzantine nodes.
Output:
    A boolean that indicates whether DolevU-T is correct on $G$.

◁ Split and add all nodes to new directed graph
$dG$ = DiGraph($\emptyset$)  ◁ Init. empty unweighted directed graph
forall $u \in V$ do
  $dG$.add_vertex($2u$)  ◁ in node
  $dG$.add_vertex($2u + 1$)  ◁ out node
  if $u \in V_t$ then $dG$.add_weighted_edge(($2u, 2u + 1$), $2f + 1$)
  else $dG$.add_weighted_edge(($2u, 2u + 1$), 1)

◁ Add directed weighted edges to directed graph
forall $(u, v) \in E$ do
  if $u \in V_t$ then $dG$.add_weighted_edge($2u + 1, 2v, 2f + 1$)
  else $dG$.add_weighted_edge($2u + 1, 2v, 1$)
  if $v \in V_t$ then $dG$.add_weighted_edge($2v + 1, 2u, 2f + 1$)
  else $dG$.add_weighted_edge($2v + 1, 2u, 1$)

◁ Find out if any nodes $u$ and $v$ can communicate reliably
forall $u \in V$ do
  forall $v \in V$ s.t. $u < v$ do
    ◁ a) Are $u$ and $v$ connected by an undirected trusted path?
    if ($u \notin V_t \land v \notin V_t \land v \in \Gamma_{f(G)}(u)$)
        or ($u \in V_t \land v \notin V_t \land v \in \Gamma_{g(G,f(G),u)}(u)$)
        or ($u \notin V_t \land v \in V_t \land u \in \Gamma_{g(G,f(G),v)}(v)$)
        or ($u \in V_t \land v \in V_t \land u \in \Gamma_{g(G,g(G,f(G),u),v)}(v)$) then
      continue

    ◁ b) Otherwise, do they have enough vertex-disjoint paths?
    if not isTrusted(u) then
      forall $w \in \Gamma_G(u)$ do
        $dG$.change_edge_weight($2u + 1, 2w, 2f + 1$)

    if max_flow($dG, 2u + 1, 2v$) < $2f+1$ then
      return FALSE

    if not isTrusted(u) then
      forall $w \in \Gamma_G(u)$ do
        $dG$.change_edge_weight($2u + 1, 2w, 1$)

return TRUE
```

Algorithm 6 Verification of `DolevU-T`'s *RC-Validity* on a graph based on graph simplification and connectivity measurement (`DolevU-T-Verif`, method 2).

```
Inputs:
    $G = (V, E)$: undirected network topology.
    $V_t \subset V$: set of trusted nodes in $G$.
    $f$: max. number of Byzantine nodes.
Output:
    A boolean that indicates whether DolevU-T is correct on $G$.

compute $f(G)$
forall $(u, v) \in V \setminus V_t$ and $u < v$ do  ◁ Pairs of untrusted nodes
    if $\kappa_{f(G)}(u, v) < 2f{+}1$ and $u \notin \Gamma_{f(G)}(v)$ then
        return FALSE

forall $u_t \in V_t$ do
    compute $G_{u_t} = g(G, f(G), u_t)$
    forall $v \in V \setminus V_t$ do  ◁ Pairs of trusted-untrusted nodes
        if $\kappa_{G_{u_t}}(u_t, v) < 2f{+}1$ and $u_t \notin \Gamma_{G_{u_t}}(v)$ then
            return FALSE
    forall $v_t \in V_t \setminus \{u_t\}$ s.t. $u_t < v_t$ do  ◁ Pairs of trusted nodes
        compute $G_{u_t,v_t} = g(G, G_{u_t}, v_t)$
        if $\kappa_{G_{u_t,v_t}}(u_t, v_t) < 2f{+}1$ and $u_t \notin \Gamma_{G_{u_t,v_t}}(v_t)$ then
            return FALSE

return TRUE
```

B.2 Complexity of the first correctness verification method for `DolevU-T`– Max-flow on transformed graph (Alg. 5)

The algorithm first requires (i) computing $f(G)$ once, which has complexity $\mathcal{O}(|V|)$, (ii) computing $g(G, f(G), u_t)$ for each trusted node u_t, which has complexity $\mathcal{O}(|V_t|{\cdot}|V|)$, and computing $g(G, g(G, f(G), v_t), u_t)$ for each pair of trusted nodes (u_t, v_t), which has complexity $\mathcal{O}(|V_t|^2{\cdot}|V|$.

The graph dG has $2|V|$ vertices and contains less than $(2|V|)(2|V| - 1) + 2|V| = 2|V|^2$ edges. The max flow between any two nodes in dG is bounded by $(2f{+}1){\cdot}2(|V| - 2 + 1) = (2f{+}1){\cdot}2(|V| - 1)$. Computing the max flow in $dgUV$, therefore, is $\mathcal{O}(8{\cdot}f{\cdot}|V|^3)$, which has to be done at most for all pairs of nodes that are not neighbors in G or connected through a path of trusted nodes. The complexity of this procedure is bounded by $\mathcal{O}(8{\cdot}f{\cdot}|V|^5)$.

B.3 Complexity of the second correctness verification method for `DolevU-T`– Eliminating unnecessary trusted nodes and checking for $2f{+}1$ vertex-connectivity (Alg. 6)

Complexity Analysis. We analyze the complexity of the main steps of Alg. 6 in the following.

Identifying the edges of $f(G)$ (Alg. 6, l. 7) requires computing connected components, which can be done using a union-find data structure [20]. This data structure is initialized with sets that each contain one vertex of $G{=}(V, E)$ and is modified by considering all edges of the trusted nodes in E and merging the sets that contain their extremities. Let $E_t \in E$ contain the edges of G's trusted nodes. Intuitively, a set in the final union-find data structure contains the nodes that can communicate using a (possibly empty) path of trusted nodes. Computing the union-find data structure in a simple way and computing the edges of $f(G)$ based on it has a linear complexity $\mathcal{O}(|V|)$.

The neighbors of a node in $f(G)$ are its untrusted neighbors in G and the untrusted nodes in the union-find sets that contain at least one of its trusted neighbors in G. Determining the neighbors of all (untrusted) nodes in $f(G)$ based on the union-find data structure therefore has linear complexity $\mathcal{O}(|V|)$.

The number of edge-disjoint paths between two nodes is equal to the maximum flow between them when all edges have unitary capacity (Menger's theorem [32]). To compute the lowest maximum flow between two nodes in graph $f(G)=(V_0, E_0)$ one can use the classical Ford-Fulkerson algorithm [18] by setting all edge capacities to 1. Computing the maximum flow between two nodes in $f(G)$ has a complexity bounded by $\mathcal{O}(|E_0|\cdot\text{maxFlow}(f(G)))$, where $\text{maxFlow}(f(G))$ is the maximum flow in $f(G)$. Evaluating whether two given untrusted nodes can communicate reliably in $f(G)$ (Alg. 6, l. 9) therefore has overall complexity $\mathcal{O}(|V\backslash V_t|^2\cdot\text{maxFlow}(f(G)))$. Because the number of distinct pairs of edges in $f(G)$ is $\frac{|V\backslash V_t|((|V|\backslash V_t|-1)}{2} = \mathcal{O}(|V\backslash V_t|^2)$, and because the maximum flow in $f(G)$ is lower than $|V\backslash V_t|-2$, an upper-bound complexity to check that all pairs of untrusted nodes have a sufficient flow is therefore $\mathcal{O}(|V\backslash V_t|^5)$.

Adding a trusted node u_t in $f(G)$ to compute $G_{u_t}=g(G, f(G), u_t)$ (Alg. 6, l. 12) requires merging some connected components of $f(G)$ by considering u_t's edges ($\mathcal{O}(|V|)$ cost), and recomputing the neighbors of each node in G_{u_t} by going through these connected components ($\mathcal{O}(|V|)$ cost). These steps need to be executed for every trusted node, which results in $\mathcal{O}(|V_t|\cdot|V|)$ complexity.

Computing the vertex connectivity between two nodes in G_{u_t} (Alg. 6, l. 14) has complexity $\mathcal{O}(|V\backslash V_t|^3)$. This step is executed $|V_t|\cdot|V\backslash V_t|$ times (all pairs of trusted-untrusted nodes), which leads to an overall complexity of $\mathcal{O}(|V_t|\cdot|V\backslash V_t|^4)$.

Building G_{u_t,v_t} from G_{u_t} (Alg. 6, l. 17) has linear complexity $\mathcal{O}(|V|)$. Building such graphs is repeated $\mathcal{O}(|V_t|^2)$ times, for a complexity $\mathcal{O}(|V_t|^2\cdot|V|$.

Evaluating the connectivity between two nodes in G_{u_t,v_t} (Alg. 6, l. 18) has complexity $\mathcal{O}(|V\backslash V_t|^3)$ (using Ford-Fulkerson's algorithm [18]). This step is repeated $\mathcal{O}(|V_t|^2)$ times, which leads to an overall complexity of $\mathcal{O}(|V_t|^2\cdot|V\backslash V_t|^3)$.

The overall complexity of Alg. 6 is bounded by the sum of all terms that appear inside boxes in this section:

$$\mathcal{O}(2|V| + |V\backslash V_t|^5 + |V_t|\cdot|V| + |V_t|\cdot|V\backslash V_t|^4 + |V_t|^2\cdot|V| + |V_t|^2\cdot|V\backslash V_t|^3).$$

When $|V_t|$ is small, $|V\backslash V_t|^5 \approx |V|^5$ dominates. When $|V_t|$ is large, then it is $|V_t|^2\cdot|V| \approx |V|^3$ that dominates.

C Intuition: Leveraging Trusted Nodes in the Authenticated Links Model

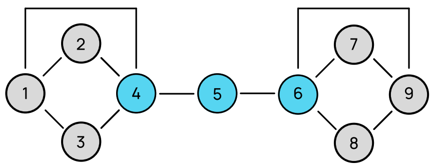

Figure 3 A network of 9 non-authenticated nodes (gray) that includes 3 trusted nodes (blue) where `DolevU-T` enforces reliable communication while `DolevU` does not.

Fig. 3 illustrates a network of non-authenticated nodes that contains 9 nodes, including 3 trusted (i.e., nodes 4, 5, and 6). In this network, reliable communication would be ensured by `DolevU-T` with at most one faulty node (i.e., $f=1$), but not by `DolevU`.

To motivate `DolevU-T`, it is interesting to observe how a message broadcast by node 1 could be authenticated by node 9, assuming that node 1 is correct. The message would first eventually be authenticated by node 4 (directly or through both untrusted nodes 2 and 3).

As node 4 is trusted, it would then vouch for its authenticity to node 5. Later on, node 5 would do the same with node 6, and node 6 would then directly disseminate the message to all its neighbors, including node 9. Since node 6 is trusted, node 9 is able to authenticate the message.

This example illustrates the fact that an uninterrupted path of trusted nodes is equivalent to a reliable link, which `DolevU-T` leverages. In particular, a message that only goes through trusted nodes is authenticated. However, `DolevU-T` also relies on the most general observation that trusted nodes can be removed from any position in the paths that `DolevU` manipulates to decide whether a message can be delivered.

D Counter-example: in `DualRC`, a node must keep forwarding signatures even after it delivers using dissemination paths

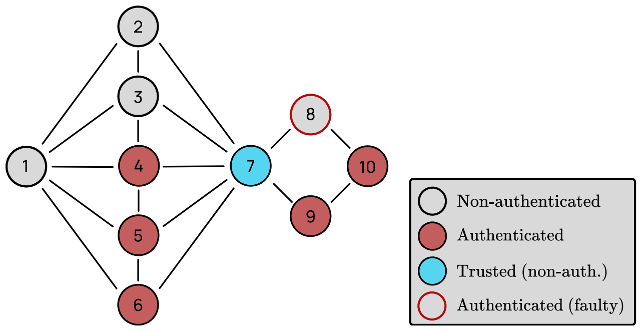

Figure 4 Example of a network where not forwarding signatures after delivering a message based on dissemination paths would prevent some nodes from authenticating it.

Without knowing the network topology, it is generally not possible for a node that delivers a message based on message paths to stop forwarding all signatures it might receive later. Fig. 4 shows a graph where doing so can prevent a correct node from authenticating a message from another correct node. In this example, nodes 1, 2, 3, and 7 are not authenticated and, therefore, do not manipulate signatures (but can relay them) and authenticate messages only based on their dissemination paths. Nodes 4, 5, 6, 8, 9 and 10 are authenticated. Additionally, 7 is trusted, and 8 is faulty. Node 1 aims at sending a message m to node 10. Let us consider the following execution:

- Message m arrives at nodes 2 and 3 that both deliver the message and forward it to their neighbors, including node 7.
- Message m arrives at node 7 through node 2 with an empty path. At this moment, node 7 is not able to authenticate m using only path [2], but it forwards m with a modified path.
- Message m arrives at node 7 through node 3 with an empty path. Node 7 authenticates m using paths [2] and [3], and forwards it with an empty path to nodes 8 and 9. After delivering m, node 7 ignores all messages it might receive from nodes 4, 5, and 6 that contain their signature on m.
- Node 8 is faulty and does not forward m further.
- Node 9 forwards m with an empty path to node 10, which cannot authenticate it based solely on [9] and subsequently never authenticates m.

Instead, if 7 keeps forwarding the signatures on m it receives after delivering it based on paths, then 10 would receive the signatures of nodes 4, 5, and 6 and would authenticate m.

▶ Remark. Note that nodes do not have to keep forwarding paths after they have delivered a message using signatures. Indeed, upon delivering a message based on signatures, it is sufficient for a node to forward an empty path to all its neighbors and ignore the paths it might receive afterwards [10].