A Cryptographic Study of Secure Internet Measurement Technical Report

Sharon Goldberg, David Xiao, Boaz Barak, and Jennifer Rexford
Princeton University

March 5, 2007

Abstract

Mechanisms for measuring data-path quality and identifying locations where packets were dropped are crucial for informing routing decisions and enforcing network accountability. If such mechanisms are to be reliable, they must be designed to prevent ASes from 'gaming' measurements to their advantage (*e.g.* by hiding packet loss or by blaming packet loss on innocent ASes). In this paper, we explore mechanisms for accurately *detecting* and *localizing* packet loss events on a data path in the presence of both benign loss (congestion, link failure) and active adversaries (ASes motivated by malice or greed). We do not advocate a specific network architecture. Instead, we use rigorous techniques from theoretical cryptography to present new protocols and negative results that can guide the placement of measurement and security mechanisms in future networks.

Our major contributions are: (1) Negative results that prove that any detection or localization mechanism requires secret keys, cryptography and storage at every participating node. (2) *Pepper Probing* and *Salt Probing*, two efficient protocols for accurate end-to-end detection of packet loss on a path, even in the presence of adversaries. (3) A new protocol for accurately localizing packet loss to specific links along a path, even in the presence of adversaries.

This technical report contains proofs, definitions, and other results in support of our short paper, "Measuring Path Quality in the Presence of Adversaries: The Role of Cryptography in Network Accountability".

Contents

1 Introduction

The Internet's best-effort service model provides neither *a priori* guarantees of packet delivery nor *post hoc* information about the fate of dropped packets. However, tools for measuring data-path quality and identifying locations where packets were dropped are crucial for informing routing decisions at the edge of the network (*e.g.* in source routing, intelligent route control, and multipath routing). Moreover, [19] recently argued that the Internet industry's resistance to evolution *cannot* be overcome without an accountability framework that measures path quality and then holds ISPs responsible for poor path performance. The combination of an accountability framework and tools for intelligent routing could counter the growing trend of commoditization on the Internet by giving ISPs an economic incentive to upgrade their networks in order to attract customers [8, 19, 1]. In this paper we explore methods that empower the edge of the network to measure data path quality. Because the network layer is typically responsible for making routing decisions, we focus specifically on tools that source edge networks (*i.e.* stub ASes or edge routers) can use to perform the following two types of measurements: *fault detection (FD)* to detect faults on a data path, and *fault localization (FL)* to localize the links at which the faults occurred. We say that a fault occurs when a packet sent by a source edge network fails to arrive unaltered at its destination edge network within a reasonable amount of time.

The presence of adversaries: The design of robust measurement schemes is complicated by the presence of parties on the network that have a *strong incentive to bias path quality measurements*. Consider these natural scenarios:

- ISPs may provide poor service to selected classes of traffic in order to cut costs. ISPs have an economic incentive to prevent their customers from taking their business elsewhere or from demanding accountability for service level agreement (SLA) violations. As such, greedy ISPs may attempt to bias measurements to prevent edge networks from detecting degraded path quality, or even to blame degraded service on an innocent ISP.
- A malicious router or AS, corrupted by a remote attacker or disgruntled network operator, may advertise routes through himself so that he can selectively block or tamper with certain flows. For example, a remote attacker may tamper with packets containing information about a corporation's stock price, sent by a website like CNN.com. The hacked router has a strong incentive to bias measurements so that his malicious behavior can continue undetected, or is attributed to an innocent router.
- A router may 'benignly' drop packets due to malfunction, misconfiguration or excessive congestion. For example, an MTU (Maximum Transmission Unit) mismatch along the path might cause a router to drop large packets, while continuing to forward small packets (*e.g.* from ping and traceroute). Rather than making assumptions about the nature of packet loss events, we can instead assume that packet loss occurs in the "worst possible way", *i.e.* according to a pattern that introduces bias in our measurements.

As such, we avoid making *ad hoc* assumptions about the good behaviour of ASes or routers on the data path. Instead, *we assume that a source edge-network can trust only the destination edge-network at the end of the data path she is measuring, while the intermediate nodes on the path may adversarially affect her measurements*. We focus on finding efficient protocols and minimal resource requirements for fault detection and localization; to avoid burdening our schemes with extra machinery, our model does not consider traditional security issues like preventing faults, providing confidentiality, or protecting against traffic analysis or denial-of-service attacks.

Accountability, Security and Measurement: Our work lies at the intersection of the literature on network accountability, data-plane security, and path-quality measurement. The literature on accountability [1, 19] focuses on ASes' economic motivations for making decisions, but it does not explicitly consider mechanisms to prevent ASes from 'gaming' an accountability framework to their economic advantage. On the other hand, much of the work [25, 2, 4, 14, 24, 21] on data-plane security implicitly assumes that faults caused by adversarial nodes should be detected on a *per-packet* basis (*i.e.* when

just one packet is dropped [25, 2, 4, 14]). Here we consider benign faults (*i.e.* congestion, link or node failures) alongside adversarial behaviour, which makes our models simultaneously more realistic and more difficult to analyze. Moreover, in addition to considering the per-packet approach, we also explore *statistical* approaches for estimating *average fault rate* (*i.e.* the fraction of packets that experienced faults) on the data path. Finally, our approach can be seen as a secure analogue of the statistical path-quality measurement techniques designed by the Internet measurement community [9, 10, 20, 15, 28] for the adversarial setting. Our work is most closely related to that of Stealth Probing [3], a statistical fault detection protocol designed for a similar adversarial setting.

Techniques from theoretical cryptography: In contrast to previous work, *in this paper we use a rigorous theoretical framework to explore a wider space of solutions for the adversarial setting; we consider both per-packet and statistical approaches for performing fault detection (FD) and fault localization (FL).* In fact, by working within the framework of theoretical cryptography to precisely define security for our adversarial setting, we have exposed security vulnerabilities of other FD and FL protocols [24, 4, 1, 10, 29] (we discuss these in the body and footnotes of the paper). Furthermore, our rigorous approach has allowed us to *prove* the security of our FD and FL measurement protocols, and to use the techniques of [16, 17, 22] to present a set of *negative results* that determine the minimum resources necessary to build any FD or FL protocol that is robust to adversaries. Though our security definitions consider active adversaries with extensive powers (Section 2, 3.2 and 4.2), many of our negative results hold even in a weaker adversarial setting (see Sections 3.3 and 4.3). We begin by introducing the reader to Alice, a source edge network, (edge router or stub AS), who sends data packets to Bob, a destination edge network. Eve, the adversary, occupies some subset of intermediate nodes on the path. *In this paper, a node can be either an AS or an individual router.*

Results on fault detection (FD): We present negative results (Section 3.3) that prove that *any* secure per-packet or statistical FD protocol requires (1) a key infrastructure, (2) cryptographic operations at Alice/Bob, and (3) dedicated storage at Alice. Our results imply that any scheme that does not make use of these resources *cannot* be secure in the adversarial setting. Next, we present a simple secure per-packet FD protocol, *Per-Packet Ack*, and two novel secure statistical FD protocols, **Pepper Probing** and **Salt Probing**. In Per-Packet Ack, Bob acknowledges each packet that Alice sends with an authenticated ack packet (Section 3.4). Pepper Probing and Salt Probing are efficient protocols that allow Alice to estimate the average fault rate, up to an arbitrary level of accuracy, without requiring any modifications to Alice's traffic. In Pepper Probing (Section 3.6), Alice and Bob sample packets in a coordinated manner with a cryptographic hash function, similar to the approach of Trajectory Sampling [10]. Salt Probing (Section 3.7) extends Pepper Probing to the public-key setting via a timing strategy similar to that of TESLA [26].

Results on fault localization (FL): We present a set of negative results that prove that *any* secure per-packet FL protocol requires all the resources required for secure per-packet FD and in addition, each intermediate node must (1) share keys with Alice, (2) perform cryptographic operations, and (3) keep dedicated storage (Section 4.3). Next, we limit ourselves to the setting of source routing with symmetric paths and present an **Optimistic Protocol** for per-packet FL (Section 4.4), and a **composition** technique for constructing a secure statistical FL protocol by having each intermediate node simultaneously run Pepper or Salt Probing to the destination edge network (Section 4.5). Our FL protocols can guide future designs of FL protocols outside the source-routing setting.

Technical highlights: This paper is a rigorous theoretical exploration of solution space for FD and FL; rather than advocating a specific network architecture, we present new protocols and negative results that can be used to guide the placement of measurement and security functions in future networks (Section 5). Our protocols leverage a variety of useful cryptographic techniques, including: (1) using randomly-selected *salt* values to run symmetric cryptographic operations in the public-key setting (Section 3.7), (2) using *onion reports* to prevent an adversary from blaming bad behaviour on an innocent node (Section 4.4), and (3) using a *composition* technique to construct a statistical FL protocol out of statistical FD protocols (Section 4.5). Moreover, while some of our proofs follow quite simply from our security definitions, many of our proofs are more involved (*e.g.* the necessity of cryptography for FD and FL). *This paper contains full proofs and definitions in support of short paper [11], which contains a high-level*

overview of our results.

2 Our setting

Goals and non-goals: When considering path quality we focus on both *availability* (path delivers packets to the correct destination) and *integrity* (packets arrive unaltered). Therefore, we say that a fault occurs when a packet sent by Alice fails to arrive unmodified at Bob within a reasonable amount of time. Our techniques can be extended to obtain finer measurements, such as calculating the average round-trip time (RTT); however, to simplify the presentation we do not discuss RTT measurements here. In this paper, we do *not* study techniques to *prevent* an adversary from adding packets to the data path or from (selectively or fully) causing faults, nor do we consider techniques that guarantee confidentiality or protect against traffic-analysis attacks. Moreover, our protocols are *not* designed to *diagnose* the cause of a fault. In the adversarial setting it is extremely difficult to accurately distinguish between benign and adversarial behaviour because an adversary can always drop packets in a way that 'looks like congestion'. As such, we do not distinguish between faults caused by benign causes (*e.g.* congestion, node failure), malicious behaviour, or even increased congestion on a link during a denial-of-service (DoS) attack. Finally, for lack of space, we do *not* explicitly address the problem of *protecting* our FD and FL protocols themselves from DoS attacks (*e.g.* an adversary crashes an FD monitor by flooding it with messages that require verification of a digital signature); however, standard rate-limiting techniques can mitigate these attacks.

Adversary model: We consider both benign faults (due to congestion or node failure) and malicious faults (caused by an active adversary). We assume that Alice cannot trust any intermediate node along the data path, except for Bob. One (or more) nodes on the data path may be controlled by an active adversary, Eve, with extensive powers: she may add, drop, or modify traffic on the data path; she may observe the traffic on the data path and use any observations, including timing information, to learn how to bias Alice's measurements; she may attempt to blame her own malicious behaviour on innocent nodes, *e.g.* by simulating the behavior of the system during past instances of congestion-related loss, as in Section 3.3; she may share information with other adversarial nodes on the data path or collude with those nodes (*e.g.* by tunnelling packets).

Per-packet vs. statistical approaches: In per-packet schemes, Alice learns if a fault occurred for *each packet* that she sends to Bob. To reduce the overhead of per-packet schemes, we also consider statistical schemes, where Alice instead estimates the average fault rate on the data path. We believe that statistical schemes are sufficient for most situations; because it is natural that some packets are dropped due to congestion, we argue that the network layer neither cares nor expects to know the fate of every single packet sent. On the other hand, if the network layer wants to detect small transient problems which selectively harm specific end-hosts (*i.e.* drops of one or two TCP SYN packets), then per-packet approaches are more appropriate.

Resources and Evaluation Criteria: We shall evaluate our protocols based on how efficiently they use the following five resources: We prefer protocols that minimize (1) *communication overhead*, (2) *computation* of cryptographic operations, and (3) use of dedicated *storage* in the router. Furthermore, the difficulty of building up and maintaining a (4) *key infrastructure* for the Internet is well known. As such, we prefer schemes that require few keys; public-key schemes, where each node has only a single key, are preferred over schemes where each pair of nodes share a symmetric secret key. Finally, we prefer protocols that (5) *do not affect the router's internal packet-processing path.* In fact, all our protocols do not modify any packets sent by the source edge-network, so that they can implemented in a monitor located off the critical path in the router. This approach has the additional advantage of minimizing latency in the router, not increasing packet size, and allowing Alice to turn measurements on and off without having to coordinate with Bob.

3 Fault Detection

Secure fault detection (FD) is an end-to-end technique that allows Alice to detect whether or not her traffic arrived unaltered at Bob, in the presence of adversaries on the data path. In this section, we start by defining security for per-packet FD, present a set of negative results that shows the resources required for any secure FD protocol, and then present a simple secure per-packet FD protocol. We then define security for statistical FD and present two novel statistical FD protocols, Pepper Probing and Salt Probing.

3.1 Fault Detection: Definitions and Notations

First we give the basic syntactic properties shared in common by all the definitions. A symmetric fault detection scheme consists of five algorithms, $\mathsf{Init}, R_A, R_B, \mathsf{Decode}, \mathsf{FaultFunction}$, which together tell Alice and Bob how to decide transmit data packets, how to recover the data from the protocol, and how to judge the faultiness of the link.

The function Init is used to create some secret auxiliary information, which Alice and Bob know but not the adversary. That is, $(\mathsf{aux}_A, \mathsf{aux}_B) = \mathsf{Init}(x)$ for a uniformly random x, and aux_A is given to Alice and aux_B is given to Bob. We do not discuss how Alice and Bob may agree upon the auxiliary in secret; the interested reader may refer to *e.g.* [12, 13] for ideas. Intuitively, this auxiliary information will provide Alice and Bob with some keys, *e.g.* a secret symmetric key or a public/private key pair.

The algorithms R_A, R_B are used by Alice and Bob respectively to encode and transmit their data. They may be interactive, *i.e.* R_A, R_B engage in several rounds of communication in order to transmit each data packet d, or they may be non-interactive, *i.e.* R_A consists of taking the data and generating a single packet, and R_B consists of taking the received packet and generating a single response message.

We will only be concerned with protocols where each data packet causes R_A, R_B to interact, without any dependence on future data. For example, this rules out protocols that wait to see two data packets d_1, d_2 and "multiplex" them together. We believe this is reasonable because the traffic that a router processes is highly unpredictable and protocols that do not transmit packets immediately may incur unreasonable amounts of delay. (Furthermore, in practice, many responses R_B generates (*e.g.* acks) may be batched together and sent in a single packet.)

We call the communication associated with transmitting a single packet d the *exchange* associated with packet d. Let $\langle R_A, R_B \rangle(d)$ denote the transcript between Alice and Bob generated when for the exchange transmitting d.

The algorithm Decode is used to retrieve the message from the exchange transcript, *i.e.* $d = \mathsf{Decode}(\mathsf{aux}_B, W_d^B, \langle R_A, R_B \rangle(d))$ where aux_B is the auxiliary information of Bob and W_d^B is the randomness Bob used to perform exchange d with Alice. Naturally we will require our systems to retrieve the data correctly when no adversary is present, *i.e.* $d' = d$ when $\langle R_A, R_B \rangle(d)$ is transcript where no adversary was present.

The algorithm $\mathsf{FaultFunction}$ is used to decide when an adversary was active during the security game. The $\mathsf{FaultFunction}$ function takes as input the auxiliary information of Alice aux_A, the history of the game H and Alice's private randomness W, and outputs $\mathsf{FaultFunction}(\mathsf{aux}_A, H, W)$, which will have a different format depending on whether we care about per-packet security or statistical security. (The format is specified in each of the definitions below.)

The normal execution of the protocol involves using Init to set up the auxiliary information of Alice and Bob, using R_A, R_B transmit the data from Alice to Bob and Bob using Decode to retrieve the data from the transcript, and finally Alice using $\mathsf{FaultFunction}$ to decide how faulty the game was.

We let $\Delta(X, Y)$ denote the *statistical distance* between two distributions X and Y [12].

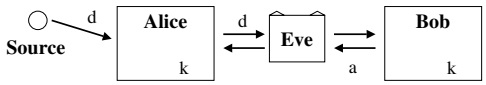

Figure 1: FD security game.

3.2 Fault Detection: Security Definition

We will define the behavior we want from the algorithms we described in the previous section. First we require that Bob can always recover the data correctly from the transcript.

Definition 3.1 (Non-triviality)**.** A per-packet fault detection scheme is *non-trivial* if for any aux_B, W_d^B, $\langle R_A, R_B\rangle(d)$ (generated according to W_d^B), we have that

$$\mathsf{Decode}(\mathsf{aux}_B, W_d^B, \langle R_A, R_B\rangle(d)) = d$$

In theoretical cryptography, game-based definitions are used to obtain precise guarantees of security [12]. We now present a game-based definition for secure per-packet FD, consisting of a description of the game setting, followed by correctness and security conditions.

Definition 3.2. The *fault detection game* for a fault detection scheme (Init, R_A, R_B, Decode, $\mathsf{FaultFunction}$), versus an adversary (i.e. an algorithm) Adv is defined as the following. Adv is given access to three oracles, a Source oracle and oracles computing R_A, R_B, which contain their respective parts of $(\mathsf{aux}_A, \mathsf{aux}_B) = \mathsf{Init}(x)$ for a random x. Each algorithm and each oracle keeps local clocks that may be used for computing time-outs or timing the arrival of packets. We assume that the oracles simulate a constant latency delay between each link.[1]

We use the 'Source' entity in Figure 1 to model the end-hosts that generate the data packets d that Alice sends Bob. When Adv queries Source a data packet d is generated and the exchange for d between R_A, R_B is initiated. The method of generating d will be depend on whether we wish to consider strong or weak security (see Definition 3.4 and Definition 3.3).

Adv is allowed to make queries to R_A and R_B oracles at will. During the execution of the game we keep track of two lists: $\mathcal{L}_{\mathsf{Sent}}$ which is a list of all the d that were generated by Source, while $\mathcal{L}_{\mathsf{Received}}$ is a list of all the d that were successfully recovered by Bob using the procedure Decode.

We'll use two different Source oracles for the fault detection game in Definition 3.2 to give weak and strong definitions. The weak version will require that the adversary win against any pre-determined message distribution, while the strong version will allow the adversary to choose the messages.

Definition 3.3. By *weak fault detection game* we mean the fault detection game of Definition 3.2 where the Source oracle generates messages d sampled independently each time according to some fixed distribution $\mathcal{D}$, subject to the condition that the collision probability of $\mathcal{D}$ is negligible (*i.e.* the probability that Source will generate identical packets is negligible). [2]

Definition 3.4. By *strong fault detection game* we mean the fault detection game of Definition 3.2 where the Source oracle generates messages given by the adversary, *i.e.* Adv specifies to Source the message d to be sent in an exchange, subject to the restriction that all messages d are unique.

[1] We do not assume any relationship between the time it takes to perform computation and the time it takes to transmit packets. This is because an adversary may be willing to invest enormous resources into his computational power and so may be able to compute faster than we expect; or, contrapositively, we cannot base our security on the assumption that the adversary is much slower than the communication links.

[2] We make this assumption to prevent Eve from trivially winning the FD game using a *replay attack*; suppose Source sends the same data packet d twice. The first time, Eve forwards d to Bob and gets valid ack a. The second time she *drops* d but responds to Alice with the valid ack a. In practice, we can prevent replay attacks by timestamping Bob's acks with an expiry time, such that no repeated packets are sent, (*e.g.* due to natural entropy in packet contents, TCP sequence numbers, and IP ID fields) for the duration of the time interval for which Bob's acks are valid.

p:	Fraction of packets acknowledged by Bob.
α:	False alarm threshold. Alice avoids raising an alarm when the fault rate is below α.
β:	Detection threshold. Alice raises an alarm when the fault rate exceed β.

Table 1: Parameters for statistical FD.

We will use the weak definition in our negative results and the strong definition in our protocol constructions. Doing this gives us the strongest possible statements; the protocols we present are secure against even strong adversaries, while our negative results apply to a broad class of protocols (even those where the adversary has no control of the data sent by end hosts).

3.2.1 Per-Packet Definition

We begin with the per-packet definition, since it is similar to the definition of classical cryptographic primitives such as message authentication codes and autheticated login systems.

In per-packet security, the output of $\mathsf{FaultFunction}(\cdot)$ is a vector of T entries, one for each exchange, where each entry takes value in $\{\mathsf{Fault}, \mathsf{NoFault}\}$.

Definition 3.5. We say a fault detection scheme is weak (resp. strong) per-packet secure if no efficient adversary Eve playing the weak (resp. strong) fault detection game can, with non-negligible probability, cause there to exist a faulty exchange that Alice does not detect. That is, the protocol satisfies the following two conditions. The **correctness condition** says that, for every exchange d in which Eve does not drop or tamper with any messages, we have that $\mathsf{FaultFunction}(\mathsf{aux}_A, H, W)_d = \mathsf{NoFault}$. The **security condition** says that no efficient Eve can cause there to exist an exchange such that data does not arrive unaltered at Bob, and yet Alice does not detect that the data was tampered with or dropped. Formally, for all efficient Eve playing the weak (resp. strong) fault detection game, we have that,

$$\Pr[\exists d \in \mathcal{L}_{\mathsf{Sent}} \setminus \mathcal{L}_{\mathsf{Received}} \; s.t. \; \mathsf{FaultFunction}(\mathsf{aux}_A, H, W)_d = \mathsf{NoFault}] \leq \varepsilon$$

for $\varepsilon = \varepsilon(n)$ a negligible function of the security parameter.

3.2.2 Statistical Definition

We argue that for many applications, Alice is more interested in determining if the average fault rate (*i.e.* the fractions of packets sent by Alice for which faults occurred) than determining the fate of each individual packet. As such, we can use statistical methods, similar to those used by the Internet measurement community, to reduce the overhead required in per-packet FD protocols. In statistical security Alice again will output Fault or $\mathsf{NoFault}$ for a group of many exchanges, which we call an *interval* of exchanges, rather than for each single exchange individually.

In statistical security, the $\mathsf{FaultFunction}(\cdot)$ function outputs a vector whose length is the number of intervals occuring in the game. For each interval j, $\mathsf{FaultFunction}(\cdot)_j \in \{\mathsf{Fault}, \mathsf{NoFault}\}$.

Definition 3.6 ((α, β)-Statistical FD Security)**.** A fault detection scheme is weak (resp. strong) $(\alpha, \beta, \delta, T_0)$-statistically secure (where $\alpha < \beta$ and T_0 is a function of α, β, δ), if for any adversary playing the weak (resp. strong) fault detection game, both the *correctness* and *security* conditions (defined below) hold. In the statistical setting, we divide the exchanges into intervals of T_0 consecutive exchanges. Let $\kappa(u)$ be the fault rate for the u'th interval. At the end of the interval, Alice either outputs Fault(if she decides the fault rate exceeds β) or $\mathsf{NoFault}$(if she decides that the fault rate did not exceed α.)
Correctness condition: Intuitively, an FD protocol is correct if Alice outputs $\mathsf{NoFault}$ at the end of

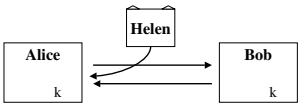

Figure 2: Helen.

an interval where less than an $\kappa(u)$-fraction of data sent by Alice did not arrive at Bob for $\kappa(u) < \alpha$, and no other messages, (*e.g.* data packets, acks) were tampered with or dropped for the remaining $1 - \kappa(u)$ exchanges.

More formally, consider an interval u where Bob fails to recover the data sent during each faulty exchange. In other words, consider only adversaries that cause fault by preventing data packets from reaching Bob; we exclude adversaries that cause faults by letting data through to Bob, but tamper with other messages sent during an exchange, *e.g.* dropping the acks sent back to Alice. We say that an FD protocol is correct if, for any such interval u and conditioned on all previous intervals, the probability that the fault rate is $\kappa(u) \leq \alpha$ and Alice outputs Fault is bounded by $\delta + \varepsilon(n)$.

Security condition: The security condition is satisfied if, for any efficient adversary Eve playing the weak (resp. strong) fault detection game, for any interval j, conditioned on all previous intervals, the probability that the fault rate $\kappa_j > \beta$ and Alice outputs NoFault is bounded by $\delta + \varepsilon(n)$.

Duration of an interval: Notice that our definition explicitly puts a lower bound on T, number of exchanges for that the game must be played for before Alice decides whether or not a fault occured. This bound on T is *essential* because a statistical scheme only measures behaviour on average; Alice can only have high confidence in her measurements when her sample size is sufficiently large.

3.2.3 Properties of Security Definition

We now discuss some properties of our security definition.

Monotonicity: Our security definition is *monotone* in that it protects not only against adversaries like Eve in Figure 1, but also against adversaries like Helen in Figure Figure 2 (a node *off the data path monitored by Alice* who attempts to trick Alice into detecting faults on the path to Bob so that Alice will switch her traffic to a path through Helen). We assume that Helen can only *add* packets to the data path between Alice and Bob. Observe that Helen could potentially trick Alice into detecting a fault (when no fault occurred) by sending Alice invalid ack packets spoofed to look like they came from Bob. However, our definition protects against attacks by Helen because the definition is *monotone*: as long as Alice receives a valid ack for every packet she sends, she will never declare a fault, *even if she receives additional nonsense packets*.

Congestion: Our definition does not explicitly model congestion. FD is an end-to-end measurement; as such, it is sufficient to detect the total fault rate over an entire path, without distinguishing between faults caused by normal congestion or adversarial behaviour.

Faulty forward or reverse path? Collecting one-way measurements of a path is notoriously difficult, since a sender cannot easily differentiate between faults on the forward path (from Alice to Bob) and faults on the reverse path (from Bob to Alice). As such, we define our FD protocols so that Alice always obtains a conservative estimate, *i.e.* a lower bound, on the fault rate on her forward path. This definition works best in the setting of *symmetric paths*, where the forward and reverse paths are identical, since here Eve has no incentive to drop the acks on the reverse path. (If she does, she simply makes the path look worse). In contrast, in the setting of *asymmetric paths*, it is useful to distinguish between faults that occur on the forward path and faults that occur on the reverse path. In this setting, Eve occupying only the reverse path may have an incentive to drop acks, perhaps to confuse Alice into thinking that the forward path is faulty.

We argue that any FD protocol that distinguishes between faults on the forward vs the reverse path must also include a coordinated path-switching mechanism between Alice and Bob, even in the benign

setting. To see why, observe that Alice cannot distinguish between (a) the forward path failing and (b) the reverse path failing. In both situations, Alice cannot communicate with Bob. However, to learn why communication failed, in case (a) the forward path must be switched, while in case (b) the reverse path must be switched. Because path-switching mechanisms are outside our scope, in this paper we do *not* construct FD protocols that explicitly distinguish between faults on the forward vs the reverse path. While we leave this interesting problem to future work, we note that such FD protocols are still subject to our negative results, and they could be designed using our simpler (Per-Packet Ack, Pepper and Salt Probing) FD protocols as building blocks.

3.3 Fault Detection: Negative Results

Here we show that *any* FD scheme that is secure according to the per-packet definition in Sections 3.2 (1) requires shared keys between Alice and Bob, (2) requires Alice/Bob to perform some cryptographic operations,[3] and (3) requires Alice to modify her memory for each packet that she monitors. Furthermore, while our results about the necessity of keys and cryptography rely on the assumption that Eve actively forges acks, our result about the necessity of storage holds even when Eve's adversarial powers are limited to dropping packets. Our negative results imply that the resource overhead of our Per-Packet-Ack, Pepper and Salt Probing protocols (Sections 3.4, 3.6 and 3.7) are unavoidable in the sense that we could not have designed them without keys, cryptography and storage.

3.3.1 Keys are Necessary for Fault Detection

Here we show that Alice and Bob need some secret *correlated* information or else schemes cannot be secure.

Proposition 3.7. *For any fault detection scheme, if* aux_A *and* aux_B *are independent, or if it is easy to guess* aux_B*, then the scheme is not secure (in any sense!).*

Proof. We prove this in the contrapositive. Assume that $\mathsf{aux}_A, \mathsf{aux}_B$ are independent so that Alice and Bob have no shared secret key. It follows that Eve knows everything that Bob knows, so that Eve can simply sample aux_B for herself and drop all the data that Alice sends to Bob, but respond in the way that Bob would respond. Therefore, since the sampled aux_B is distributed identically to the aux_B of honest Bob, Eve convinces Alice that nothing is wrong, and the FD scheme cannot be secure.

Furthermore, if it is easy to guess aux_B, then clearly Eve can win again whenever she guesses aux_B correctly by simulating Bob's responses to Alice and while dropping all the packets that Alice sends to Bob. ■

Thus, for the remainder of the paper we will assume that there is indeed shared secret information between Alice and Bob. In particular, we will assume that they either have shared secret keys or a public/private key pair.

3.3.2 Cryptography is Necessary for Fault Detection

We now prove that the keys shared by Alice and Bob must be used in a 'cryptographically-strong' manner. We now show that any secure per-packet FD is *at least as complex* as a a secure keyed identification scheme (KIS). KIS is well-studied cryptographic primitive that is know to be at least as complex as symmetric-key encryption. (In Appendix B provide an equivalent argument by reducing secure FD directly to one-way functions.) In a **secure keyed identification scheme (KIS)** [12] Alice and Bob share a secret key, and Alice ascertains Bob's identity by asking him to respond to a challenge (*e.g.* solve a riddle) that can only be responded to by someone who knows the secret key. (To illustrate this, Figure

[3] Listen [29] is an FD protocol that does not require Alice or Bob to perform cryptographic operations. Our results imply that Listen is insecure in our adversarial setting.

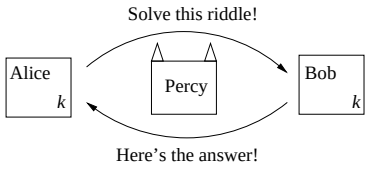

Figure 3: KIS security game.

3 we show a sample two-message KIS protocol. In practice a KIS can have a more abitrary structure, *e.g.* a four-message protocol, etc.) A KIS is secure if an impersonator who does not know the key can't respond to the challenge with better success than by just guessing a random answer. Here, we are concerned with KIS schemes that are secure against a passive adversary, Percy (the impersonator), who eavesdrops on the interactions between Alice and Bob and then tries to impersonate Bob by responding to Alice's challenge on his own

Our proof implies that secure identification is an unavoidable part of secure FD. That is, any secure FD protocol must ensure that Alice can distinguish between acks sent by Bob, and acks sent by an adversary impersonating Bob. Furthermore because KIS is known to be at least as complex as symmetric-key encryption [16, 22], our proof implies that *one cannot expect to design secure FD protocols that run blazingly faster than symmetric-key encryption protocols.*[4]

We start with the per-packet setting.

Theorem 3.8. *Any weakly-secure per-packet fault detection scheme is at least as computationally complex as a KIS that is secure against a passive adversary.*

Proof. To prove that any secure FD scheme is at least as complex as a secure KIS, we show that an arbitrary secure FD scheme can be used to construct a new KIS. The construction is simple: Alice's key in our new KIS is aux_A (from the FD scheme). Bob's key in our new KIS is aux_B (from the FD scheme). The challenge in our new KIS is Alice's algorithm R_A (run on one exchange d of the FD scheme). The correct response to the challenge in our new KIS is Bob's algorithm R_B (run on one exchange of the FD scheme). Alice accepts Bob's identification in this KIS iff Alice outputs NoFault at the end of the exchange (of the FD scheme).

To complete the proof, we must show that if the FD scheme used in the above construction is secure, then our new KIS construction is also secure. We do this in contrapostive, by showing that if there existed an efficient adversary Percy that breaks the security of this KIS construction, then Percy can be used to construct an adversary Eve that breaks the security of the FD protocol. As in Figure 4, Eve uses Percy to break the security of the FD protocol as follows: At first, when Percy wants to eavesdrop on an interaction between Alice and Bob, Eve asks the Source to generate a random packet d. Eve then lets Percy see the exchange between Alice and Bob related to packet d. Note that Percy gets to see messages that are exactly what he expects to see in the KIS security game. Next, when Percy is ready to impersonate Bob, Eve again asks the Source for a packet d, but now instead of forwarding Alice's messages pertaining to d to Bob she *drops* all communication (for this exchange) between Alice and Bob, while responding to Alice with Percy's message. The proof follows from the fact that Eve successfully tricks Alice into thinking the Bob received Alice's messages (and therefore breaks the security of the FD protocol) whenever Percy successfully responds the the challenge in the KIS (and therefore breaks the security of KIS by impersonating Bob). ■

We now extend this proof to the statistical setting.

[4]Our reduction from per-packet secure FD to secure KIS is essentially "tight" in the sense that one exchange of the FD protocol becomes one authentication session fo the KIS protocol. Showing that secure KIS implies symmetric-key encryption incurs a much larger loss of efficiency: the most straight-forward proof requires going through one-way functions. It is an open problem whether one could construct a secure symmetric-key encryption from secure KIS in a more efficient manner.

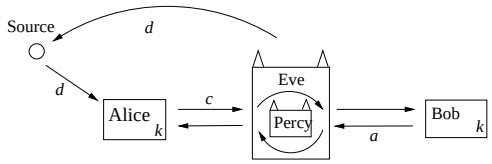

Figure 4: Reduction from FD to KIS.

Theorem 3.9. *Any weakly-secure $(\alpha, \beta, \delta, T_0)$ fault detection scheme (for T_0 a function of α, β, δ) is at least as computationally complex as a KIS that is secure against a passive adversary.*

Proof. This (contrapostive) proof is entirely identical to the proof of the per-packet case except that now, we define one interaction of the KIS as a entire *interval* (*i.e.* T_0 exchanges) of the FD protocol (instead of defining one interaction of the KIS as a single *exchange* of the FD protocol.) That is, in the per-packet case the KIS was a two-message protocol; in the statistical case, the KIS is a $2T_0$-message protocol.

Whereas before we showed that Eve can break security by dropping a packet and using Percy to impersonate Bob acks during a single exchange, Eve now breaks security by dropping packets during an entire *interval* and using Percy to impersonate Bob during the entire interval. During the impersonated interval, since Percy impersonates Bob's behaviour an interval where the the fault rate does not exceed α, Alice will not raise an alarm (during the impersonated interval) even though the true false rate exceeded β. As such, Eve again breaks the security of the FD protocol, completing the contrapositive proof. ■

3.3.3 Storage is necessary for Fault Detection

In this section, we prove that in secure FD Alice must modify storage for (almost) each packet that she samples. We start with a proof for the per-packet FD, and then extend the proof to the statistical setting.

Theorem 3.10. *In a per-packet weakly secure FD protocol, then Alice must modify storage at least once during each exchange.*

Proof. We prove this in the contrapositive. Assume that Alice had stored no information (*i.e.* does not modify her memory) for some exchange related to a packet d. It follows that Eve can simply block all communication from Alice to Bob, and Alice won't notice (because Alice immediately forgets about the packet d after she sends it), and the FD scheme cannot be secure. ■

We now extend the proof to the statistical setting, for *FD protocols based on sampling*. In an FD protocol based on sampling, Alice decides if each packet it sends requires acknowledgement from Bob or not. We say that a packet that Alice requires Bob to acknowledge is *sampled*.

Theorem 3.11. *In any $(\alpha, \beta, \delta, T_0)$-weakly secure FD protocol based on sampling, Alice must modify storage for at least $\Omega((\frac{\beta}{\beta-\alpha})^2 \ln \frac{1}{1/2+\delta})$ exchanges.*[5]

Proof. We prove this in the contrapositive. Suppose that Alice modifies storage for for less than $\Omega((\frac{\beta}{\beta-\alpha})^2 \ln \frac{1}{\frac{1}{2}+\delta})$ exchanges. Each exchange where Alice does not modify storage cannot be sampled

[5]Note that this bound is essentially independent of δ. One can obtain better dependence on δ at the cost of worse dependence on $\beta - \alpha$, namely something like $\Omega((\frac{\beta}{3\beta-\alpha})^2 \ln \frac{1}{\delta})$. This is done by a simple modification of our proof to require Eve to drop at a rate of 3β instead of β and then showing that the probability that the fault rate of the game is below β is exponentially small. However, since the dependence on δ is logarithmic anyway we prefer to focus on the stated version of the theorem.

since Alice does not even remember seeing that interval. Let $T < \Omega((\frac{\beta}{\beta-\alpha})^2 \ln \frac{1}{\frac{1}{2}+\delta})$ be the actual number of exchanges that Alice samples.

Now consider the adversary that drops packets at random with probability β. With probability 1/2 it will drop at least a β fraction of the exchanges.

Now Alice samples only T exchanges, and let $X_1, \ldots, X_T$ be whether or not the i'th exchange she samples is dropped by Eve. Notice that $\mathbb{E}[X_i] = \beta$. Now Alice's estimate of the fault rate will be at most $\frac{1}{T}\sum_{i=1}^{T} X_i$. Letting $\mu = \frac{\alpha+\beta}{2}$, this is incorrectly undetected with probability

$$\Pr[\tfrac{1}{T}X_i < \mu] > \binom{T}{\mu T}\beta^{\mu T}(1-\beta)^{\mu T}$$

which, using Stirling's approximation (or see the alternative derivation in [?]) is at least

$$\geq \tfrac{1}{T+1}2^{-D(\mu\|\beta)T}$$

Here $D(\cdot\|\cdot)$ is the relative entropy function. Looking at the Taylor expansion of $D(\cdot\|\cdot)$ with respect to μ, β, we see that for our range of parameters $D(\frac{\alpha+\beta}{2}\|\beta) \leq \frac{16(\beta-\alpha)^2}{\beta}$, and so the above is at least

$$\geq \tfrac{1}{T+1}2^{-16\frac{(\beta-\alpha)^2}{\beta}T}$$

For our choice of T, this gives us that the probability is at least

$$> 1/2 + \delta$$

So Eve's overall probability of breaking security is:

$$\Pr[\kappa > \beta \text{ and } \tfrac{1}{T}X_i < \tfrac{\alpha+\beta}{2}] > 1 - (1/2 + 1/2 - \delta) > \delta$$

■

3.4 Per-Packet-Ack: A Simple FD Protocol

Per-Packet Ack is a simple secure per-packet FD protocol, where Bob securely acknowledges receipt of every packet he sees. Despite the simplicity of the protocol, shown in Figure 5, we explain it here in some detail in order to introduce the reader to some of the cryptography we use in later parts of the paper.

The protocol is built from two cryptographic primitives: A **collision-resistant hash function (CRH)** [12], which we denote by h, is a hash function for which it is computationally infeasible to find two inputs $x \neq x'$ that map to the same output $h(x) = h(x')$. In practice h can be implemented with a function such as SHA-2 [23].

Definition 3.12 (CRH)**.** A function family $\mathcal{H} = \{h\}_k$ mapping $h_k : \{0,1\}^* \to \{0,1\}^{n'}$ for $n' < n$ is a family of collision-resistant hash functions if for all efficient algorithms A, the probability over a random $h \xleftarrow{R} \mathcal{H}$ that A can find m, m' such that $h(m) = h(m')$ is negligible, or formally

$$\Pr_{h \xleftarrow{R} \mathcal{H}}[A(h, 1^n) = (m, m') \; s.t. \; h(m) = h(m')] \leq \varepsilon_{\mathsf{crh}}$$

where $\varepsilon_{\mathsf{crh}}$ is the *CRH-advantage* and is negligible in n.

A secure deterministic **message authentication code** (MAC) [12] protects a message's integrity and authenticity by allowing a receiver, who shares a secret key k with the sender, to detect any changes made to the sender's message. A MAC is secure if no efficient adversary can forge a valid signature on any arbitrary message (with non-negligible probability). We use the notation $[m]_k$ to denote message m MAC'd with key k.

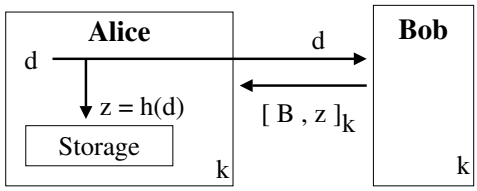

Figure 5: Per-Packet Ack protocol.

Definition 3.13. Formally, a message authentication code consists of two algorithms $(\mathsf{Sign}, \mathsf{Ver})$ where Sign is used to MAC the message, and Ver is used to verify the validity of the MAC on a message. A MAC is secure if for all efficient algorithms Fred, where Fred gets access to oracles for Sign and Ver[6], and Fred outputs a message and signature (m, σ), subject to the condition that m was not queried to Fred's Sign oracle, then

$$\Pr_k[\mathsf{Fred}^{\mathsf{Sign}_k(\cdot),\mathsf{Ver}_k(\cdot)}(1^n) = (m,\sigma)\ s.t.\ \mathsf{Ver}_k(m,\sigma) = 1] \leq \epsilon_{\mathsf{mac}}(n)$$

for a negligible function $\epsilon_{\mathsf{mac}}(n)$. We call $\epsilon_{\mathsf{mac}}(n)$ the *MAC-advantage*.

The Per-Packet-Ack protocol is a simple protocol where each packet is acknowledged with a MAC'd ack. This protocol satisfies strong per-packet security.

Definition 3.14. Init generates a truly random secret key k to be shared by Alice and Bob. We assume there is some publicly known CRH h. Alice sends each data packet d completely in the clear to Bob, and stores a packet digest[7] $z = h(d)$ along with a time-out (we say that a fault occurs if the time-out expires before Alice receives an ack for that packet). Then Bob receives the packet d, he computes the packet digest $z = h(d)$ and sends an ack of the form $a = [B, z]_k$ where B is a public unique identifier for Bob's identity. Alice removes a packet digest z from storage when she receives an acknowledgment a containing a matching packet digest z and a valid MAC. Periodically, Alice checks her storage to see if any packets awaiting acknowledgment have timed out; if so she raises an alarm and declares that a fault occurred.

Theorem 3.15. *Per-Packet-Ack is strongly per-packet secure.*

Proof. Notice that in this protocol, acks are *unambiguous*: they specifically depend on the contents of the packet d that Bob receives. It follows that if Eve wants to create a valid ack to a packet d that was dropped before it reached Bob, she either has (i) to find a collision d' in the CRH h, so that $h(d) = h(d')$, and ask Bob to generate an ack for d', or (ii) she has to forge the MAC signature on an ack $(B, h(d))$ without Bob's help. By the definition of CRH, the probability (i) that the adversary finds d, d' such that $h(d) = h(d')$ is at most $\varepsilon_{\mathsf{crh}}$.

Conditioned on event (i) not happening, we can also show that (ii) occurs with probability at most $\varepsilon_{\mathsf{mac}}$ by reducing the Per-Packet-Ack Protocol to MAC. To do this, observe that if we had an Eve breaking the security of the Per-Packet-ACK protocol, we could construct an adversary, Fred, that forges signatures. (Fred can simulate the FD game to Eve as follows: He simulates the Source and Alice by generating random, unique messages d and passing them on to Eve, and then simulates Bob by computing the packet digest $z = h(d)$, getting his Sign oracle to generate the ack $a = [B, z]_k$ by querying the oracle on (B, z), and then passing a along to Eve. Then Fred can use Eve to win the MAC game by outputting the ack a the Eve generated during the exchange where Eve forged the MAC on some packet Eve dropped. During the exchange where Eve drops a packet, Fred need not query his Sign oracle on the packet, so that if Eve wins the FD game, then Fred will have a MAC on a unique message that was never queried to the Sign oracle.)

[6] But Fred does not know the key used by the Sign and Ver oracles

[7] Packet digests must be computed only on *immutable* fields of the packet, that are unchanged as the packet traverses the data path. See [10] for immutable fields in an IP packet.

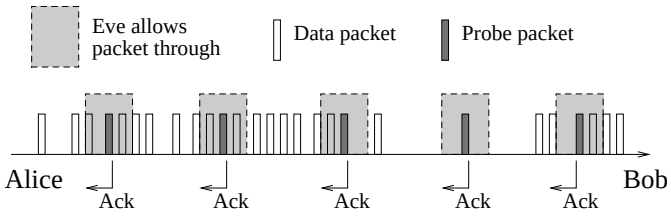

Figure 6: Attack on active measurement.

Thus the overall advantage of any efficient adversary is at most $\varepsilon_{\mathsf{crh}} + \varepsilon_{\mathsf{mac}}$ which is negligible, so Per-Packet-Ack is secure. ■

Remark 3.16 (Ambiguousness)**.** A protocol is unambiguous if distinct data packets have distinct acknowledgements. Consider for example a scheme is one in which Bob sends "Got a probe" each time he receives a probe. Intuitively, this scheme should be insecure, because an adversary can drop all of Alice's messages to Bob and convince Alice to accept the route by sending Alice "Got a probe" for every packet he drops. This a similar type of attack is possible if the protocol simply sends back a protocol in which Bob sends back a count of the number of packets or bits received (*e.g.* as suggested in one version of Fatih [21]); namely, the adversary can drop all of Alice's packets, and simply send Bob a bunch of nonsense packets such the the count of the number of packets that Bob receives is identical to the number of packets or bits sent by Alice.

One way to improve the communication overhead of Per-Packet Ack protocol is to "batch together" many acknowledgments into a single super-ack packet. We note however that the batching operation should preserve the unambiguousness property of the Per-Packet-ACK protocol. The obvious way to do this is to send many acks $a_1, a_2, .., a_{50}$ in a single 'super-ack' packet.

Moreover, because packets can arrive at Bob out of order, any batching operation must either reconstruct the exact same order of packets as sent Alice or must unaffected the order of the packets. The former option is very heavy-weight since it means that some ordering information must be added to the data, while the latter option means essentially that Bob has to include a separate packet digest for each individual packet he receives in in the super-ack anyway, so that he does not save much in the way computation. Furthermore, batching acks together may increase the storage requirements at Bob and Alice, since now Alice has to store her data for a longer of period of time (RTT plus time it takes to create a batch).

3.5 Why use Statistical Fault Detection based on Sampling?

The per-packet FD protocol of Section 3.4 required acks and memory modifications every packet sent. To reduce the high overhead required for per-packet protocols, we now study techniques that require only an accurate measurement of the average fault rate. We will focus on schemes that use sampling.

Sampling: Instead of performing FD on every single packet, as in the Per-Packet Ack protocol that we present in Section 3.4, in sampling schemes we randomly select a p fraction of the packets to monitor in order to obtain an estimate of average behavior. Our estimate becomes accurate if we obtain a sufficient number of samples. We let *probing schemes* denote statistical FD schemes that use sampling, and *probes* denote packets that require acks, and $p < 1$ denote the fraction of packets sent by Alice that are acknowledged by Bob. Security in the statistical setting guarantees that *Eve cannot bias Alice's estimate of the average fault rate.*

Probe indistinguishability and the trouble with active measurement: Any probing scheme that is secure in the adversarial setting requires a property called probe indistinguishability [3, 1]. That is, to prevent Eve from biasing Alice's measurements by treating probe packets preferentially (while, say, dropping all other packets), *Eve must not be able to distinguish probe packets from non-probe packets.* In active probing, Alice injects specially crafted probe packets into her data traffic stream according to

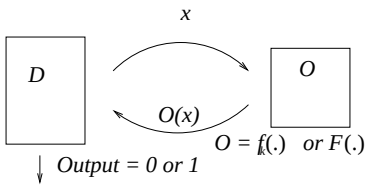

Figure 7: PRF distinguishing game.

some arrival process, and performs FD on probe packets only (*e.g.* ping, traceroute, and the approaches of [20, 28, 15]). Active probing is undesirable for a number of reasons; for example, injecting additional special probe packets into the system interfers with measurement [5] and increases traffic on the network. Furthermore, to prevent trivial attacks on probe indistinguishability, active probes must also be designed to 'blend in' with data traffic (*e.g.* by padding and randomizing probe packet contents and length, or by encrypting the entire traffic + probe scheme, as in Stealth Probing [3].) Furthermore, we now show how use timing information can be used to break probe indistinguishability, even when the entire traffic + probe scheme is encrypted.

To illustrate these timing attacks, consider an extreme example where probes are injected according to a periodic arrival process and Bob sends an acknowledgement for each probe packet as in Figure 6. Eve can easily learn how to distinguish probes from existing traffic by observing when Bob sends ack packets. Eve can then drop all packets sent outside a small time window around the beginning of the probe period. Notice that Eve can easily discover the period of the probe arrival process begins by observing when Bob sends his acknowledgement packets, thus learning how to distinguish probes from non-probes. Then, Eve can attack by dropping all the packets that are sent outside a small time window around the beginning of the probing period. [8] Furthermore, even when active probes arrive according to a randomized arrival process (*e.g.* a Poisson process), the statistics of the probe arrival process are likely to be different from those of the data traffic arrival process, and this information can be used by a clever adversary to distinguish probe from data traffic. As such, in the remainder of this paper we avoid schemes based on active measurement, and instead focus on *passive sampling* protocols that sample from existing traffic.

3.6 Pepper Probing: Symmetric-Key Statistical FD

Pepper Probing extends the Per-Packet Ack protocol of Section 3.4 to the statistical setting by allowing Alice and Bob to sample packets in a coordinated way by computing a 'cryptographic hash' over packet contents (similar to Trajectory Sampling [10]'s approach).

Psuedo-Random Functions (PRFs): The protocol relies on the properties of pseudorandom functions. A PRF [12] is a keyed function that cannot be distinguished by any computationally-efficient algorithm from a *truly random function*. (A random function $F : \{0,1\}^n \to \{0,1\}^n$ can be thought of as the following process: for each one of its possible 2^n inputs x, choose a random n-bit string to be $F(x)$. This means that we need $2^n \cdot n$ coins to choose a random function.) A pseudorandom function collection is a function that can be described with only n bits but is indistinguishable from a random function. We say that the collection is efficiently computable if the mapping $(k, x) \mapsto f_k(x)$ is computable in polynomial time.

Definition 3.17. A pseudorandom function collection $\mathcal{F} = \{f_k\}$ is a collection of efficiently computable functions that are computationally indistinguishable from a random function F. Formally, we say that $\mathcal{F}$ is a PRF family if for all efficient oracle algorithms D with oracle access to $\mathcal{O}$ containing either $f_k(\cdot)$

[8] This attack can be avoided if Bob sends out acknowledgements at fixed time intervals that are independent of the probe arrival process. However, by doing this Alice loses the ability to measure packet delay through the network.

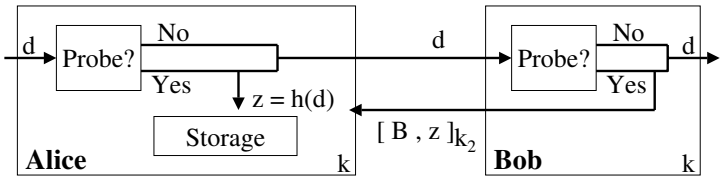

Figure 8: Pepper Probing.

for a random $k \in \{0,1\}^n$ (as shown in Figure 7) or $F(\cdot)$ for a truly random function F[9], it holds that

$$\left| \Pr_k[D^{f_k(\cdot)} = 1] - \Pr_F[D^{F(\cdot)} = 1] \right| \leq \epsilon_{\mathsf{prf}}(n)$$

for negligible function $\epsilon_{\mathsf{prf}}(n)$. We call ϵ_{prf} the *PRF-advantage*.

In practice, the high-throughput PRF we require for Pepper Probing can be efficiently and securely implemented in hardware with pipelined AES in CBC-mode or with a cryptographic hash function [23]. However, the oft-suggested CRC function with a secret modulus $f_k(x) = x \mod k$ should not be used because it is not a PRF! To see why, observe that the CRC is a linear function that can easily be distinguished from a random function. Consider a distinguisher D interacting with it's oracle $\mathcal{O}$ as in Figure 7. D first asks its oracle for $\mathcal{O}(x)$, and then asks for $\mathcal{O}(x+1)$. To distinguish between the CRC and the random function, D checks if $\mathcal{O}(x+1) = \mathcal{O}(x)+1$. If it is, D decides that $\mathcal{O} = f_k$, and otherwise D decides that $\mathcal{O} = F$. ■

3.6.1 Description of Pepper Probing

We are now ready to describe the Pepper Probing protocol, shown in Figure 8.

Definition 3.18 (Pepper Probing)**.** As shown in Figure 8, Alice and Bob share a secret $k = (k_1, k_2)$. For each packet sent, they use k_1 to compute a function Probe that determines whether or not a packet d is a probe and should therefore be digested $z = h(d)$, stored, and acknowledged. To acknowledge a probe, Bob sends Alice an ack that is MAC'd using k_2 to key a secure MAC, as in Per-Packet Ack (Section 3.4). The Probe function is implemented using a PRF. which we can think of as a keyed hash function (keyed here with k_1) that takes in an input (here, the data packet d), and outputs a number $f_{k_1}(d) \in \{1, \ldots, 2^n\}$. We let

$$\begin{array}{ll} \mathsf{Probe}_k(x) = \text{Yes} & \text{if } \frac{f_{k_1}(d)}{2^n} < p \\ \mathsf{Probe}_k(x) = \text{No} & \text{otherwise} \end{array} \tag{3.1}$$

For each interval u, Alice stores each probe packet (*i.e.* each packet d such that $\mathsf{Probe}_{k_1}(d) = \text{Yes}$) in a table $\mathcal{L}_{\mathsf{ack}}(u)$. Upon receiving ack $a = [z]_{k_{\mathsf{mac}}}$: Alice verifies that a is correctly MAC'd, and if so, removes z from her table $\mathcal{L}_{\mathsf{ack}}(u)$. To obtain an $(\alpha, \beta, \delta, T_0)$-secure protocol, the $\mathsf{FaultFunction}(\cdot)$ function outputs Fault for the u'th interval when $\frac{|\mathcal{L}_{\mathsf{ack}}(u)|}{pT} > \frac{\alpha+\beta}{2}$, while it outputs $\mathsf{NoFault}$ when $\frac{|\mathcal{L}_{\mathsf{ack}}(u)|}{pT} < \frac{\alpha+\beta}{2}$.

3.6.2 Security of Pepper Probing

Theorem 3.19. *Pepper Probing is a strongly $(\alpha, \beta, \delta, T)$-statistically secure fault detection scheme for any $0 < \alpha < \beta < 1$ and $0 < \delta < 1$ where probing intervals last for $T_0 = O(\frac{\beta}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ exchanges.*

Proof. For the purpose of this proof, we refer to game G, as the strongly-secure FD game using a statistical winning condition, played for the the Pepper Probing protocol.

Next, we divide the probability that Eve manages to bias Alice's measurement into two parts: either (i) Eve produces an ack to a dropped packet by finding a collision in the CRH forging a MAC (ii) otherwise

[9] D does not know if $\mathcal{O}$ contains $f_k(\cdot)$ or $F(\cdot)$.

(*e.g.* she violated probe indistinguishability and then made sure to drop only non-probe packets). Let E denote the event that (i) is true. Thus, we want to show that for any efficient Eve, the probability Eve winning the strong fault detection game in the $(\alpha, \beta, \delta, T_0)$- statistical sense, is

$$\begin{aligned}\Pr[\mathsf{Eve} \text{ wins game}] = &\Pr[\mathsf{Eve} \text{ wins game } G \text{ and } \mathsf{Eve} \text{ forges an ack to a dropped packet}] \\ &+ \Pr[\mathsf{Eve} \text{ wins game } G \text{ and } \mathsf{Eve} \text{ fails to forges an ack for any dropped packet}] \\ &= \Pr[\mathsf{Eve} \text{ wins game } G \cap E] + \Pr[\mathsf{Eve} \text{ wins game } G \cap \overline{E}] \end{aligned} \tag{3.2}$$

which is bounded by a negligible function of the security parameter n.

We can show that

$$\Pr[\mathsf{Eve} \text{ wins game } G \cap E] < \epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{mac}} \tag{3.3}$$

using an argument identical to the one we used in the proof of security for Per-Packet-Ack.

We continue the proof by showing that if the Probe function uses a PRF, then Pepper Probing is strongly $(\alpha, \beta, \delta, T_0)$-statistically secure. We do this using a contrapositive argument. We start by assuming that Pepper Probing is insecure, so that for some α, β, δ and $T_0 = O(\frac{\beta}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$

$$\Pr[\mathsf{Eve} \text{ wins game } G \cap \overline{E}] \geq \delta + \tilde{\varepsilon} \tag{3.4}$$

where $\tilde{\varepsilon}$ is a *non-negligible* function of n. We shall show that that this implies that the PRF is insecure (*i.e.* that we can construct a distinguisher D that can distinguish between an PRF and a Random Function, see Definition 3.17).

Recall that we denote by game G the strongly secure FD game using Pepper Probing protocol, which makes use of a PRF in the Probe function. Now, consider game G', which is identical to game G, except that now, the strongly secure FD game is played for a FD protocol that is just like Pepper Probing *except that the PRF in the* Probe *function is replaced with a random function.*

Now in game G', because Alice and Bob use a truly random function decide which packets become probes, it follows from the definition of the Probe function (Equation 3.1) any packet is probe with truly random probability p. Furthermore, since was assume that all packets sent by the Source oracle are unique (thus preventing Eve from violating probe indistinguishability by observing which packets where non-probes in the past, and then dropping those packets when reappear), it follows that Eve cannot predict which packet is a probe with probability better than p. In other words, in Game G' probes are truly indistinguishable from non-probes. Finally, note that the uniqueness of packets and the properties of random functions imply that in game G' each packet is *independently* sampled (*i.e.* is designated as a probe) with probability p. Letting $\kappa(u)$ denote the true fault rate of interval u and $H_{1,\ldots,u-1}$ denote the history of intervals 1 through $u-1$, it follows that, for *every interval* u,

$$\Pr[\mathsf{Eve} \text{ wins game } G' \cap \overline{E}] \leq \Pr[\tfrac{|\mathcal{L}_{\mathsf{ack}}(u)|}{pT} \leq \tfrac{\alpha+\beta}{2} \mid (\kappa(u) > \beta), H_{1,\ldots,u-1}, \overline{E}]$$

where we used the fact that $\Pr[\mathsf{Eve} \text{ fails to forges an ack to any dropped packet}] \leq 1$. Now, because a truly random function gives us truly independent samples, and since Eve did not forge any acks, it follows that $|\mathcal{L}_{\mathsf{ack}}(u)|$ is distributed as a binomial random variable $X \sim B(\kappa(u)T_0, p)$ (*i.e.* a binomial random variable sampling with probability p out of T elements), so that

$$\leq \Pr[\tfrac{X}{pT} \leq \tfrac{\alpha+\beta}{2} \mid \kappa(u) \leq \alpha]$$

Using the fact that $\mathbb{E}[X] = \kappa(u)T_0$ and $\kappa(u) > \beta$, we can derive that the above is bounded by

$$\leq \Pr[\tfrac{X}{T} - \beta p < (\alpha - \beta)p/2]$$

Using a Chernoff bound [12] (which tells us the estimated fault rate should be close to the true fault rate if the number of exchanges sampled $T_0 = O(\frac{\beta}{(\beta-\alpha)^2} \ln \frac{1}{\delta})$ is sufficiently large), we conclude that

$$\Pr[\mathsf{Eve} \text{ wins game } G' \cap \overline{E}] < e^{-\frac{(\beta-\alpha)^2 p}{12\beta} T_0} < \delta$$

Now using Inequality 3.4 and the inequality above, we have that

$$\Pr[\mathsf{Eve} \text{ wins game } G \cap \overline{E}] - \Pr[\mathsf{Eve} \text{ wins game } G' \cap \overline{E}] > (\delta + \tilde{\varepsilon}) - \delta = \tilde{\varepsilon} \tag{3.5}$$

We now claim that the Eve in Equation 3.5 can be used to construct an efficient distinguisher D that breaks the security of the PRF as follows: D simply simulates to Eve the entire strong fault detection game using Pepper Probing, replacing calls of the Probe function to the PRF or random function with calls to the oracle $\mathcal{O}$ in the PRF distinguishing game. At the end of Eve's execution, F checks if Eve wins and if event $\overline{E}$ happens (*i.e.* that Eve does not forge any acks). Then, the distinguisher D outputs 1 iff Eve wins game $G \cap \overline{E}$. From Equation 3.5 it follows that

$$\Pr_k[D^{f_k(\cdot)} = 1] - \Pr_F[D^{F(\cdot)} = 1] > \tilde{\varepsilon} > \epsilon_{\mathsf{prf}}(n)$$

since we assume $\tilde{\varepsilon}$ is non-negligible, which breaks the security of the PRF, completing our contrapositive argument.

From Equation 3.2, Inequality 3.3, and the contrapositive argument above, we now know that

$$\Pr[\mathsf{Eve} \text{ wins game } G] < \delta + \epsilon_{\mathsf{mac}} + \epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{prf}}$$

and since $\epsilon_{\mathsf{mac}}, \epsilon_{\mathsf{crh}}, \epsilon_{\mathsf{prf}}$ are each negligible, it follows that Pepper Probing is $(\alpha, \beta, \delta, T_0)$-secure as long as each interval lasts for $T_0 = O(\frac{\beta}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ exchanges. ■

A similar Chernoff bound can be used to show the Pepper Probing is correct.

Theorem 3.20. *Pepper Probing is correct for $(\alpha, \beta, \delta, T)$ for any $0 < \alpha < \beta < 1$ and $0 < \delta < 1$ where probing intervals last for $T_0 = O(\frac{\alpha}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ exchanges.*

Now since $\alpha < \beta$, Theorem 3.19 and Theorem 3.19 imply that Pepper Probing is both correct and strongly $(\alpha, \beta, \delta, T)$-statistically secure when probing intervals last for at least $T_0 = O(\frac{\beta}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ exchanges.

Remark 3.21. Security fails without pseudo-randomness: Suppose that the Probe function of Equation 3.1 was implemented using a CRC keyed with a secret modulus, as in [10], instead of a PRF. Model the CRC function as $f_k(x) = x \mod k$, and consider the following attack: Eve starts by observing the interaction between Alice and Bob, and recording a list of packets that were not acknowledged by Bob. Then, whenever she sees a new packet that is within a small additive distance of old packet that was not acknowledged, she drops the packet. Thus, Eve can drop non-probe packets with high probability, and she can bias Alice's estimate F below the true fault rate. This attack is possible because the CRC does not use its 'secret key' in 'cryptographically-strong' manner (*c.f.,* our result on the necessity of cryptography, Section 3.3).

3.6.3 Efficiency of Pepper Probing

For a reasonable set of parameters, say overhead $p = 0.02$, false-alarm threshold $\alpha = 0.005$, and detection threshold $\beta = 0.01$, and $\delta = .01$ it follows from the statement of Theorem 3.19 (or see Table 2) that Pepper Probing is (α, β)-secure when the protocol runs for at least $T > 1.3 \times 10^6$ packets. If each packet digest is 128 bits long, it follows that Alice needs about $pT \times 128 = 3.3$ Mbits of storage. (While Alice requires this much storage for each Bob network, she can choose to run Pepper Probing with only a small number of Bob networks at a time by ignoring all the acks sent by all other Bob networks.) Pepper Probing incurs only a small communication overhead of p (due only to to acks; Alice's traffic is unmodified). Consider now the Stealth Probing protocol of [3], another secure statistical FD protocol which also uses passive sampling and authenticated acks for a p-fraction of packets. Whereas Stealth Probing requires Alice to authenticate and encrypt all of her traffic, Pepper Probing does not require any modifications to Alice's traffic.

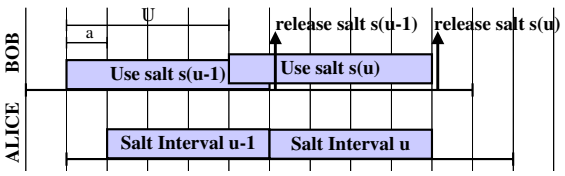

Figure 9: Timing for Salt Probing.

3.7 Salt Probing: Public-Key Statistical FD

While Pepper Probing requires pairwise symmetric keys between each Alice-Bob pair, Salt Probing requires fewer keys by operating in the public-key setting. While known public-key cryptographic primitives are too computationally expensive to execute at line rate, we bypass this problem by using techniques reminiscent of TESLA[10] [26], so that public-key operations are used very infrequently. Hence, the computational overhead of Salt Probing is almost identical to that of Pepper Probing. Like TESLA, Salt Probing requires that Alice and Bob coarsely synchronize their clocks. Time is divided into *salt intervals*. In each interval Bob uses the *same salt value* to key each probing session with each Alice network.

3.7.1 Description of Salt Probing

Setup: Alice has some local secret key k_A, which is not shared with anyone; she uses this key to verify data that she sends to Bob and then is sent back to her. Bob has a public key PK_B that is known to Alice, and he keeps secret the corresponding secret key SK_B. Alice and Bob also know a fixed time constant a, which approximately equals 1.5 round trip times (RTT) (*e.g.* $a = 150$ ms). Before the protocol begins, *Alice synchronizes her clock so that it lags Bob's clock by at most a seconds* as follows: At time t_A (on Alice's clock) Alice sends Bob a message $[A, t_A]_{k_A}$ MAC'd using her local secret key. Bob receives this message at time t_B (on Bob's clock) and responds with digitally signed message $[B, t_B, [A, t_A]_{k_A}]_{SK_B}$. (Note that a **digital signature** performs the same function as a MAC, but in the public-key setting; here the key SK used to sign a message is secret while the key PK used for verification is publicly known.) Alice will accept Bob's time t_B as long as his message is validly signed, contains a valid copy of Alice's message $[A, t_A]_{k_A}$, and arrives within a seconds of time t_A (on Alice's clock). Then, at time $t_A + a$ Alice synchronizes her clock to Bob's time t_B. If, after many attempts, Alice fails to receive a valid response to her synchronization message, then she decides the data path is faulty and raises an alarm. Synchronization happens infrequently (say, once a day).

Bob's algorithm: At the beginning of each salt interval u, Bob randomly chooses a pair of *salt* values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ that he keeps secret for the duration of the salt interval. Then, Bob runs a slightly modified version of Pepper Probing, replacing the symmetric key $k = (k_1, k_2)$ in Figure 8 and Equation 3.1 with the salt values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$. That is, during salt interval u, Bob runs the Probe function of Equation 3.1 on packet digests $z = h(d)$ using salt $\mathsf{s}_1(u)$ as a key, and for each packet that is a probe, he constructs and sends to Alice an ack of the form $[B, z, u]_{\mathsf{s}_2(u)}$, which contains a packet digest z and a salt interval number u. As shown in Figure 11, Bob reveals the salt $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ to Alice (and to all other source networks connected to him) a seconds after salt interval u ends by sending a (public-key) signed *salt release message* $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$.

Alice's algorithm: Because Alice does not know the salt for the duration of the salt interval, she is unable to run Probe 'in real time' as she sends each packet (as she did in Pepper Probing). Instead, Alice will store a packet digest for *each* of the packets that she sends to Bob as shown in Figure 10. Whenever Alice receives an ack $[B, z, u]_{\mathsf{s}_2(u)}$ from Bob, she stores the ack in her table only if the ack is tagged with the current salt interval u and a packet digest z that matches a packet digest that she has stored in her table. While on the surface it may seem that in Salt Probing Alice needs to store information about each packet she sends to Bob for the duration of a salt interval, storage requirements can be reduced without

[10] TESLA uses only symmetric-key operations (except if a PKI is used for key exchange). Salt Probing could also be adapted to TESLA's symmetric key setting by adopting TESLA's use of one-way chains [26].

packet digest	ack	Probe
$z_1 = h(d_1)$		Yes
$z_2 = h(d_2)$	$[B, z_2, u,]_{\mathsf{s}_2(u)}$	Yes
$z_3 = h(d_3)$		No
$z_4 = h(d_4)$		No
$z_5 = h(d_5)$		No
$z_6 = h(d_6)$		No
$z_7 = h(d_7)$		No
$z_8 = h(d_8)$	$[B, z_8, u]_{\mathsf{s}_2(u)}$	Yes
$z_9 = h(d_9)$		No
$z_{10} = h(d_{10})$		No
$z_{11} = h(d_{11})$	$[B, z_{11}, u]_{\mathsf{s}_2(u)}$	Yes
$z_{12} = h(d_{12})$		Yes
$z_{13} = h(d_{13})$		No

Figure 10: Alice's table after at the end of salt interval u. Here Alice observes faults during exchanges $1, 12$.

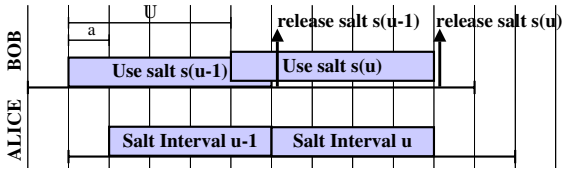

Figure 11: Overview of Timing for Salt Probing.

compromising security if Alice *independently subsamples* packets with (truly random) probability q (and then stores only the subsampled packets in memory).

If Alice fails to receive a validly signed salt release message $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$ after salt interval u ends, she concludes that her data path to Bob is faulty and raises an alarm. Otherwise, Alice uses salt $\mathsf{s}_1(u)$ (from the salt release message) to run the Probe function on the packet digests in her table, and salt $\mathsf{s}_2(u)$ to verify the acks in her table. Then, to compute an estimate F of the fault rate that occurred during the salt interval, Alice counts the fraction of exchanges for which $\mathsf{Probe}(z) =$ Yes and no valid ack is stored in her table. Finally, Alice raises an alarm if $F > \frac{\alpha+\beta}{2}$ and decides that everything was normal if $F < \frac{\alpha+\beta}{2}$.

3.7.2 Timing for Salt Probing

Need for global clock synchronization: In Salt Probing, it is extremely important that Bob releases salt $\mathsf{s}(u)$ only after Alice stops accepting acknowledgements signed by the salt $\mathsf{s}(u)$. To do this, Alice and Bob require some (coarsely) globally synchronized clocks. To see why, consider the (timing-related) attack shown in Figure 13 that could be performed when Alice and Bob's clocks are not synchronized: Suppose Bob releases the salt $\mathsf{s}(u)$ at the end of his salt interval u, and enters salt interval $u+1$. Eve will now trick Alice into thinking that Bob is still in salt interval u as follows: Eve stores the salt message instead of forwarding it Alice. Then, when Eve gets packets from Alice, instead of forwarding these packets to Bob, she uses $\mathsf{s}(u)$ to (simulate Bob by) running the Probe? function and acknowledging the appropriate packets. Eve does this for the a period of time equal to the length of a salt interval, and then forwards the salt message containing $\mathsf{s}(u)$ to Alice. Notice Alice does not realize that her packet failed to reach Bob, because she has received valid acknowledgements and salt message for salt interval u.

Clock Synchronization: As shown in Figure 12, our synchronization protocol allows Alice to syn-

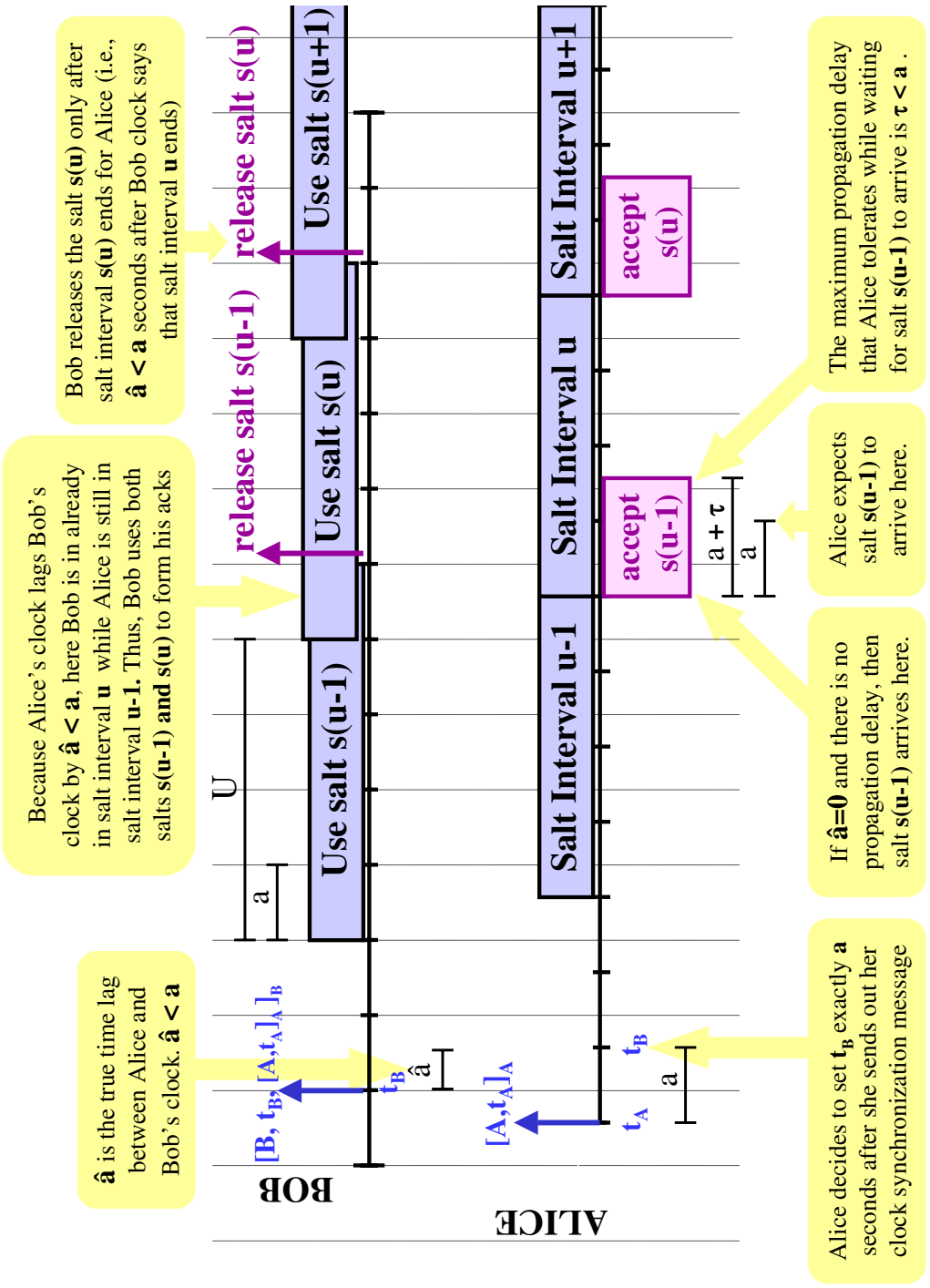

Figure 12: Detailed Overview of Timing for Salt Probing.

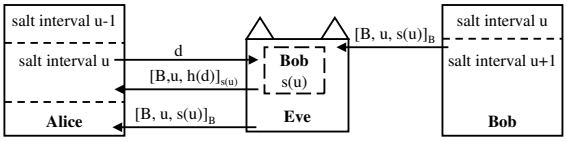

Figure 13: Attack on salt probing without global clocks.

chronize her clock to Bob's clock to within exactly $\hat{a}$ seconds, where $\hat{a} \leq a$.

Releasing salt: Bob releases the salt for interval u only after he is certain that Alice is no longer accepting acks generated using $s(u)$. Since Alice's clock lags Bob's clock by at most a seconds, Bob releases the salt at least a seconds after his clock indicates that salt interval u is over. See Figure 12.

Double salting: Note however from Figure 12, that because Alice's clock lags Bob's clock by at most a seconds, it follows that there will be period of time of length $< a$ where Alice is operating in salt interval $u-1$ while Bob has already moved into salt interval u. To remedy this, during the first a seconds of each salt interval, Bob uses *both* the salt of the current interval $\mathsf{s}(u)$ and the salt from the *previous* interval $\mathsf{s}(u-1)$ in order to create his acks.

Accepting salt: As shown in Figure 12, Alice accepts the salt $s(u)$ only after salt interval u ends (on her clock). Furthermore, since Alice knows that Bob releases the salt a seconds after the salt interval ends (on Bob's clock), Alice will somewhat 'naively' expect the salt $s(u)$ to arrive a seconds after the salt interval ends. Finally, we let τ be the maximum length of time Alice is willing to wait for the salt to arrive (before purging her storage of the table (as in Figure 10) related to salt interval u. We expect that τ is on the order of $\frac{1}{2}$RTT. Therefore, the length of time that Alice should accept the Salt is $a + \tau$ which is bounded by $2a$.

Length of the salt interval: The salt interval lasts for U seconds. To understand how to choose U, first observe that because the length of time required for 'double-salting' is a seconds, it follows that (to avoid frequent 'double-salting' or even 'triple-salting', etc.) we should have $U > a$. Futhermore, the longer U is (relative to a), the less frequently 'double-salting' is required. On the other hand, the longer the salt interval lasts, the more storage is required (see Figure 10).

Next, observe that a must be larger than 1 RTT, (because a is the length of time Alice is willing to wait before receiving a response from Bob to her synchronization message, what on average takes on 1 RTT to arrive). We add a safety factor of 50% and assume that $a \approx 1.5RTT$.

Also, from Figure 12, we observe that $\hat{a}$, the true time lag between Alice and Bob's clock is equal to a minus the one-way propagation delay that Alice's synchronization message $[A, z]_{k_A}$ experienced before it reached Bob. It follows that on average, $\hat{a} = a - \frac{1}{2}$RTT.

3.7.3 Security of Salt Probing

Proving the security of Salt Probing is very similar to proof of security for Pepper Probing.

Theorem 3.22. *Salt Probing is a strongly $(\alpha, \beta, \delta, T_0)$-statistically secure fault detection scheme for $T_0 = O(\frac{\beta}{pq(\beta-\alpha)^2} \log \frac{1}{\delta})$ sufficiently large.*

Proof. The security of Salt Probing relies on (i) Eve's inability to skew Alice's synchronization to Bob's clock beyond a seconds, (ii) Eve's inability to forge a salt release message, and (iii) Eve's inability to bias Alice's estimate of the fault rate during a salt interval.

We start by arguing that (i) Eve cannot skew Alice's synchronization to Bob's clock beyond a seconds. To see why, first observe that Alice will reject the synchronization reply $[B, t_B, [A, t_A]_{k_A}]_{SK_B}$ if Eve delays it for more than a seconds. Next, because the sync reply includes Alice's sync request $[A, t_A]_{k_A}$, which is timestamped relative to Alice's clock, it follows that the Eve cannot skew Alice's synchronization by

replaying an old synchronization reply from Bob. It follows that the only way Eve can skew Alice's synchronization to Bob is to either (a) forge Alice's MAC on a synchronization request $[A, t_A]_{k_A}$ (and then ask Bob to produce a synchronization reply to the forged synchronization request, that Eve gives to Alice after some time lag of greater that a seconds) or (b) to forge Bob's digital signature on his synchronization reply. By the security of the MAC and digital signature schemes used in Salt Probing we have that

$$\Pr[\mathsf{Eve} \text{ wins by skewing synchronization}] < \epsilon_{\mathsf{mac}} + \varepsilon_{\mathsf{sig}}$$

For the duration of salt interval u on Alice's clock, Alice will store in table (see Figure 10) packets and acks stamped with salt interval u, which she then verifies using the salt she obtains after the salt interval ends. We argue that (ii) Eve cannot trick Alice into using an incorrect salt value to verify the acks she receives unless she Eve forges the digital signature on the salt message $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$. To see why, observe that the salt release message is stamped with the salt interval u. Since by assumption Bob is honest, the salt release message stamped with interval u will always contain the salt used in interval u[11] It follows that if Eve managed to produce a salt release message stamped with interval u but containing a salt value $s'(u)$ that is different from the salt value $s(u)$ actually used by Bob during salt interval u, then she must have forged Bob's signature. (The formal reduction from Eve that breaks the security of FD game by forging a salt release message to Fred to breaks the security of a digital signature game procceeds exactly along these lines.) As such, it follows that

$$\Pr[\mathsf{Eve} \text{ wins by forging salt release}] < \varepsilon_{\mathsf{sig}}$$

Now, conditioned on Eve not skewing synchronization and Eve not forging salt release, observe that the salt $s(u)$ used to construct and verify acks during salt interval u kept secret for the duration of salt interval u on Alice's clock. (See Section 3.7.2 and Figure 12 for more details). Furthermore, at the end of the salt interval, Alice knows the correct salt value $s(u)$ that is used to verify her messages. It follows that we can use a similar argument that we used in the Theorem 3.19 to prove that (iii) Eve cannot bias Alice's measurement of the fault rate during the salt interval.

The only wrinkle in the argument here is that now Alice is subsampling packets with probability q (*i.e.* Alice is only storing packets in her table with probability q. For the remaining $1 - q$ packets she sends, she simply 'forgets' about them after she sends them.) As in Theorem 3.19, we denote by game G the strongly secure FD game using Salt Probing protocol, which makes use of a PRF in the Probe function, and Game G' as the strongly secure FD game is played for a FD protocol that is just like Salt Probing *except that the PRF in the Probe function is replaced with a random function.*

Now in game G', because Alice and Bob use a truly random function decide which packets become probes, it follows from the definition of the Probe function (Equation 3.1) any packet is probe with truly random probability p. From the properties of random functions imply that in game G' each packet is *independently* sampled by Alice (*i.e.* is designated as a probe by Alice and Bob AND is subsampled by Alice) with truly probability pq. (Recall that by definition Alice subsamples truly randomly). Using the identical argument used in Theorem 3.19, we can show that in Game G', Alice's count of the number of packets in her table (see Figure 10) that are not correctly acknowledge is is distributed as a binomial random variable $X \sim B(\kappa(u)T, pq)$ (*i.e.* a binomial random variable sampling with probability pq out of $\kappa(u)T$ elements).

Applying now applying logic similar to that used in Theorem 3.19 (replacing p with pq), we have that

$$\Pr[\mathsf{Eve} \text{ skews Alice's fault rate estimate in game } G] < e^{-\frac{(\beta-\alpha)^2 pq}{12\beta} T_0} + \epsilon_{\mathsf{mac}} + \epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{prf}}$$

which is bounded by $\delta + \epsilon_{\mathsf{mac}} + \epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{prf}}$ if we set $T_0 = O(\frac{\beta}{pq(\beta-\alpha)^2} \ln \frac{1}{\delta})$. It follows that Eve cannot skew Alice's fault rate estimate during an interval lasting for at least T_0 exchanges.

[11] TESLA [26] does not assume that 'Bob' is honest. Instead, Bob commits to the salt (using a hiding and binding commitment scheme [12]) at the beginning of the salt interval, and a the end of the salt interval Bob opens his commitment.

It follows that

$$\begin{aligned}\Pr[\mathsf{Eve}\text{ wins}] \leq & \Pr[\mathsf{Eve}\text{ skews synchronization}] + \Pr[\mathsf{Eve}\text{ forges salt release}] \\ & + \Pr[\mathsf{Eve}\text{ skews Alice's fault rate estimate}] \\ \leq & (\epsilon_{\mathsf{mac}} + \epsilon_{\mathsf{sig}}) + \epsilon_{\mathsf{sig}} + (\delta + \epsilon_{\mathsf{mac}} + \epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{prf}})\end{aligned}$$

which completes the proof of security. ■

Again, a similar Chernoff bound can be used to show that Salt Probing is correct.

Theorem 3.23. *Salt Probing is correct for* $(\alpha, \beta, \delta, T)$ *for any* $0 < \alpha < \beta < 1$ *and* $0 < \delta < 1$ *where probing intervals last for* $T_0 = O(\frac{\alpha}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ *exchanges.*

Again since $\alpha < \beta$, Theorem 3.22 and Theorem 3.22 imply that Salt Probing is both correct and strongly $(\alpha, \beta, \delta, T)$-statistically secure when probing intervals last for at least $T_0 = O(\frac{\beta}{p(\beta-\alpha)^2} \log \frac{1}{\delta})$ exchanges.

3.7.4 Efficiency of Salt Probing

While on the surface it may seem that in Salt Probing Alice needs to store information about each packet she sends to Bob for the duration of a salt interval, recall that storage requirements can be reduced without compromising security if Alice *independently subsamples* packets with (truly random) probability q (doing this of course lengthens the duration of the probing session by a factor of $\frac{1}{q}$, see Theorem 3.22).

For reasonable parameters: $p = 0.02$, false-alarm threshold $\alpha = 0.005$, and detection threshold $\beta = 0.01$, it follows from Theorem 3.22 that Alice must store at least $qT = 1.3 \times 10^6$ entries in her table (Figure 10). If each packet digest $z = h(d)$ is 128 bits long, Alice must store on average about $128(1 + p) + 1 = 131$ bits for each row of her table, she requires about $qT \times 131 \approx 170$ Mbits of storage. Now, assuming that average packet is 3000 bits long, RTTs are on the order of 100 ms, and line rates are 5 Gbps, then about 10^7 packets are sent during one RTT. Then, if a salt interval lasts for about 5 RTTs, it follows that it is sufficient for Alice to subsample packets at rate $q = \frac{1.3 \times 10^6}{5 \times 10^7} \approx 2.6\%$ in order to obtain an unbiased measurement during a salt interval.

3.8 Comparison of Fault Detection Protocols

We compare Salt Probing, Pepper Probing, Per-Packet Ack and Stealth Probing [3] in Table 2.

While the Per-Packet Ack provides the strongest security guarantee, enabling Alice to detect if even a single one of her packets is dropped or modified, the scheme suffers from 100% communication overhead (even though, in practice acks can be batched together and sent in a single packet, the requirement for unambiguousness implies that the batched together ack is likely to be large because it must include information about each individual packet sent.) The Per-Packet Ack protocol is probably best suited for applications that require a high level of availability and integrity.

Stealth Probing [3] is a secure passive probing protocol, in which Alice designates a p-fraction of data packets as probes. To ensure data integrity and to prevent Eve from distinguishing probes from non-probes, the entire data stream is encrypted and MAC'd. While the Stealth Probing reduces communication overhead to p, the protocol still requires Alice to modify all the traffic she sends. Thus, Stealth Probing must be built into the router's packet processing path.

Like Stealth Probing, Pepper Probing protocol has limited communication overhead. Pepper Probing does not modify the packets sent by Alice, and thus can be implemented as a monitor off of the routers critical packet-processing path. In Pepper Probing the only cryptography that must be computed on every single packet is a PRF (keyed cryptographic hash) operation that can be efficiently and cheaply implemented in hardware. However, because Pepper Probing is designed for the symmetric key setting, it requires pair-wise security associations between every Alice-Bob pair. Furthermore, a Bob network

	Per-Packet Ack	**Pepper**	**Salt**	**Stealth** [3]
communication overhead (due to acks)	100%	p	p	p
key structure	symmetric	symmetric	public	symmetric
clock synch between Alice and Bob?	No	No	Coarse	No
modifications to Alice's traffic?	No	No	No	Yes
minimum duration of probing session, in packets	1	$\frac{64\beta}{p(\beta-\alpha)^2}$	$\frac{1}{q}\frac{64\beta}{p(\beta-\alpha)^2}$	$\frac{64\beta}{p(\beta-\alpha)^2}$
number of packet digests stored at Alice	all	p-fraction	q-fraction	p-fraction

Table 2: Comparison of $(\alpha, \beta, \gamma, T_0)$-secure FD protocols for $\delta = 0.01$.

engaging in probing with many Alice source networks will need to look up the symmetric key required to construct the appropriate ack packets for each packet he receives.

Finally, Salt Probing inherits the limited cryptographic and communication overhead of Pepper Probing (with the addition of a few, infrequent, public-key operations) while also scaling to setting of many Alice-Bob pairs. In contrast to Pepper Probing, in Salt Probing, Bob does not need to perform a key lookup for each packet he receives, since the same salt is used for each of Bob's probing sessions with each different Alice source network. Furthermore, because the protocol works in the public key setting, for a network of M edge-networks we require only M keys (rather than M^2 keys as in the symmetric setting). However, the protocol requires (coarsely) globally synchronized clocks; we presented a simple protocol for clock synchronization in Section 3.7.

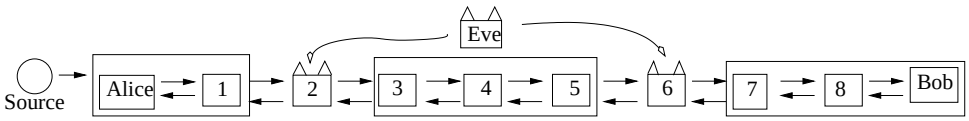

Figure 14: Fault localization security game.

4 Fault Localization

In fault localization (FL) Alice must not only detect if her path to Bob is faulty, but she must also localize the faults to a particular link (or set of links) where she suspects the faults occurred. In this section, we first define security for per-packet FL and present a set of negative results that show the resources required for *any* secure FL protocol. We present an *Optimistic Protocol* for secure per-packet FL that leverages the useful cryptographic technique of *onion reports*. After discussing security for statistical FL, we show why the most natural technique for obtaining statistical FL from statistical FD protocols (where the source runs a probing protocol with each of the intermediate nodes on the path simultaneously [24,4]) is vulnerable to a timing attack. Finally, we present a statistical FL protocol composed by having each intermediate node simultaneously run Salt or Pepper Probing with the destination network.

Localizing links, not nodes: In the adversarial setting, it is impossible for Alice to always localize a *node* controlled by Eve. To see why, refer to Figure 14, where Eve controlling R_2 can always either: (a) become unresponsive by ignoring all the packets she receives from R_1, or (b) pretend that R_3 is unresponsive by dropping all communication to and from R_3. Notice, however, that case (a) could also have been caused by R_1 refusing to send packets to R_2, and case (b) could also have been caused by R_3 becoming unresponsive. It follows that at best, a fault localization protocol can pinpoint a *link* that is adjacent to a malicious node(s), rather than the malicious node itself. As such, in Figure 14, it suffices for Alice to localize the fault in case (a) to link $(1, 2)$ and the fault in case (b) to link $(2, 3)$.

Assumptions: We model and design FL protocols for the setting of *source routing* with *symmetric paths*. However, these assumptions only make our negative results stronger; a secure FL protocol that operates in a more general setting (where Alice does not know the identity of the nodes on the path) requires at least as much overhead as a secure FL protocol designed for source routing with symmetric paths.

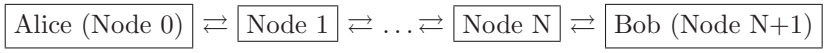

Figure 15: A path from Alice to Bob via N intermediate nodes.

4.1 Fault Localization: Definitions and notation

In this section we formally define a fault localization scheme and give justifications for our definition. We will let A, B denote Alice and Bob (i.e. the sender and receiver, respectively), We say that the direction towards Alice is "upstream" and the direction towards Bob is "downstream". We say an *exchange* is all the communication associated with sending a single data packet d.

We will often associate Alice with 0 and Bob with $N + 1$ for notational convenience. We let n be the security parameter and throughout the paper we let $\ell_i(n)$ be polynomials in n. (The ℓ_i will usually be the lengths of various parameters, and are unrelated to one another. We call them all ℓ_i simply so we don't waste letters naming all the parameter lengths.)

Definition 4.1. A fault localization scheme is a collection of the following algorithms.

- $\mathsf{Init} : \{0,1\}^n \to (\{0,1\}^{\ell_1(n)})^{N+2}$. This algorithm generates some shared initial state between the nodes. For example, this may generate shared symmetric keys or public/private key pairs for the parties. However, there is no *a priori* requirement that the shared information be used in a

cryptographic manner, and thus we choose to call this the *auxiliary information* shared by the clients rather than call them keys. Formally we will use the notation

$$(\mathsf{aux}_A, \mathsf{aux}_B, \mathsf{aux}_1, \ldots, \mathsf{aux}_N) = \mathsf{Init}(U_n)$$

If we want to prohibit trusted setup then we could set Init to be constant; however we will later show that any secure scheme must use the trusted setup in a meaningful way (and in particular Init cannot be constant).

- Algorithms $R_A, R_B, R_1, \ldots, R_N$ which take auxiliary information aux_i and can communicate amongst each other. To avoid clutter we will typically omit writing aux_i and it will be understood that R_i denotes $R_i(\mathsf{aux}_i)$. For $i \in [N]$, (R_i, R_{i-1}) can communicate and (R_i, R_{i+1}) can communicate.. In addition, R_A (Alice) communicates with some data source: in normal execution the data source will be its routing clients, while in our definition of security the data source will be controlled by the adversary.

  For a data packets d, we let $\langle R_i, R_{i+1}\rangle(d)$ denote the transcript of all messages communicated between algorithms R_i and R_{i+1} when Alice tries to send d to Bob. Note that the messages in $\langle R_i, R_{i+1}\rangle(d)$ can depend not only on R_i, R_{i+1} but also on other nodes; for example, R_i may forward messages from R_{i-1} to R_{i+1}. We call all the transcripts $\langle R_i, R_{i+1}\rangle(d)$ for $i = 0, 1, \ldots, N$ associated with sending one packet d the *exchange* of the protocol associated with d.

  We allow the algorithms R_i to be both randomized and stateful, *i.e.* their behavior can depend on uniform random coin flips and also on previously executed exchanges. We will typically not make the dependence on randomness and history explicit for clarity of notation; the appropriate notation will be introduced when necessary (*e.g.* when dealing with ideal ciphers).

  If one of the nodes is deviating from its normal behavior (i.e. it is using some algorithm V'_j instead of R_j), we will write $\langle R_i, R_{i+1}\rangle^{R_j = V'_j}(d)$ to denote the transcript in the presence of this deviant behavior.

- Decode, which takes the transcript between R_N and R_B, Bob's auxiliary information aux_B, and outputs some data.

  $$\mathsf{Decode}(\mathsf{aux}_B, \langle R_N, R_B\rangle(d)) = d'$$

  Naturally we will require that $d = d'$ if no fault occurred.

- $\mathsf{FaultFunction}$, which takes a history H of transcripts between R_A and R_1, Alice's auxiliary information aux_A, and decides where (if any) faults occurred.

  $$\mathsf{FaultFunction}(\mathsf{aux}_A, H) = F$$

  where F is a vector containing the fault information.
  For statistical security, the vector F has $N+1$ entries, one for each link, and F_i indicates the fault rate on the link between R_j and R_{j+1} during the interaction described by H.
  For per-packet security, the vector F has T entries, one for each exchange, and each entry is either a link ℓ where Alice believed that exchange was dropped/modified, or the symbol $\perp$ meaning she believes that Bob received the data for that exchange.

Remark 4.2. Our definition assumes that each packet is processed separately by the protocol: sending many packets $d_1, \ldots, d_m$ creates a transcript that is simply the concatenation of their individual transcripts (possibly with some interleaving). However, one might want to allow for an even more general definition where we can "multiplex" a bunch of packets together to send them more efficiently or more securely. In particular this would mean that the transcript for sending many packets cannot easily be decomposed into separate transcripts for each of the individual packets.

Although allowing for the more general definition would indeed capture more protocols than the definition we give above, it greatly clutters the notation and obscures the essence of the protocol. Furthermore, it is not particularly useful for this paper because the secure schemes we construct will all satisfy Definition 4.1 (i.e. they do not employ any multiplexing), and all of our impossibility proofs extend immediately to the more general definition. We will explicitly comment on this after our impossibility proof.

Property	Weak	Strong
Location of adversary	Before the game begins, the adversary picks a single node in the path that she will control.	The adversary picks an arbitrary subset of the nodes in the path that she will control *after* seeing some honest transcripts.
Power of adversary	The adversary cannot control what data is are sent from Alice to Bob; they are drawn from some fixed distribution.	The adversary decides what data Alice sends to Bob, subject to the condition that data packet are unique, *i.e.* she cannot choose the same data packet more than once.

Table 3: Weak vs Strong Security for Fault Localization

4.2 Fault Localization: Security Definition

There are several non-security related properties we want from a fault localization scheme. As usual, we require all the algorithms $\mathsf{Init}, R_A, R_B, R_1, \ldots, R_N, \mathsf{Decode}, \mathsf{FaultFunction}$ to be efficient algorithms running in poly(n). In addition, we require the following properties:

Definition 4.3 (Non-triviality)**.** A fault localization scheme is *non-trivial* if, for all possible data packets d, we have

$$\mathsf{Decode}(\mathsf{aux}_B, \langle R_N, \mathsf{Bob}\rangle(d)) = d$$

Next, we define a robustness property. Intuitively, this states that when a link is down the fault localization algorithm will locate the link precisely. We will let $R_i = \perp$ denote node i being completely unreachable, which is equivalent to link $(i-1, i)$ being down.

Definition 4.4 (Robustness)**.** Let H denote a history where in some exchange t the link $(i-1, i)$ is completely down (*i.e.* node i is unreachable). Let H' be the same history except with exchange t replaced by a non-faulty transcript. We say that a fault localization scheme is *robust* if, letting $F = \mathsf{FaultFunction}(\mathsf{aux}_A, H)$ and $F' = \mathsf{FaultFunction}(\mathsf{aux}_A, H')$ we have that $F_{(i-1,i)} = F'_{(i-1,i)} + 1$ and for all links $l \neq (i-1, i)$ we have $F_l = F'_l$.

It will turn out that all of our definitions of per-packet security, even the weakest, will imply robustness, and thus requiring that a scheme be robust does not harm the generality of our results for per-packet schemes. On the other hand, robustness does not necessarily hold in a statistical setting; indeed in our composition protocols robustness is only approximately achieved (only for those exchanges that are sampled).

Next we define security. Intuitively, there are two ways an adversary, Eve, can break the security of the scheme. First, she can try to evade detection, *i.e.* Eve prevents Bob from correctly recovering a packet, and she convinces Alice that either no fault occurred or, if Alice knows that a fault occurred, that Eve did not cause the fault. Next, Eve can break the scheme using a false negative; *i.e.* Eve does not prevent Bob from recovering a packet, but she still convinces Alice that a fault occurred somewhere, but not where Eve is located. We will formalize this intuitive definition in the following security games.

We will give a sequence of security definitions, with varying degrees of strength. In the rest of the paper, we will show impossibility results for the weaker definitions (showing that there is some lower bound on the complexity of the algorithms R_i and in particular a lower bound on the amount of cryptography they have to do), and construct schemes that satisfy the stronger definitions.

Before formally stating our definitions we give an intuitive idea of the "strong" versus "weak" characterizations that we will give. These properties are listed in Table 3. In general we will try to prove negative results for the weak formulations of the properties and construct schemes that satisfy the strong versions.

Definition 4.5 (Unreliable-link weak security game for FL)**.** For this weak definition of the security game, we require that, for any efficient adversary Eve trying to disrupt a significant fraction of the data Alice transmits to Bob, Alice will be able to localize a link adjacent to Eve with high probability.

The game setting: We will view Eve as sitting at node i, fixed ahead of time, on the path between Alice and Bob, and she gets to interact with the nodes upstream and downstream from her in a black-box way. We call these the Upstream and Downstream oracles. Eve is also given access to a Source oracle, which will transmit data to Alice to be sent to Bob. This is illustrated in Figure 16.

Oracle interactions: for this weak definition, the only thing Eve can do with the Source oracle is to ask it to send a message; we assume that Source generates a random message d from a fixed distribution with negligible collision probability, and sends it to the Upstream oracle, which then simulates Alice trying to send d to Bob. Upstream forwards this response back to Eve. This interaction is depicted in Figure 17.

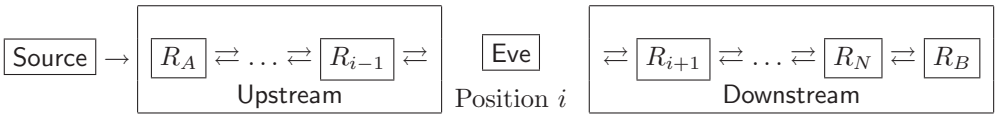

Figure 16: Game setup for weak Eve.

Eve can send Upstream an input, and Upstream will simulate $R_{i-1}, \ldots, R_1, R_A$ on this input, and return any appropriate response to Eve, and likewise for Downstream which simulates $R_{i+1}, \ldots, R_N, R_B$. These are illustrated in Figure 18.

Congestion: in this game, in each exchange, each link will be down with some constant probability $\rho > 0$. The Source, Upstream, Downstream oracles are defined so that in each exchange the oracles will flip a ρ-biased coin for each link, and will consider that link to be down for that exchange if the coin says 1 or behave normally in that link otherwise.[12] It is important to emphasize that the oracles are responsible for simulating congestion: from Eve's point of view it has perfect links to Upstream, Downstream, but each time Eve sends a packet to the Upstream, Downstream oracles they will randomly drop it with probability ρ, and each time they send a response to Eve they will also drop it with probability ρ.

Bookkeeping: In addition to simulating R_j for $j \neq i$, the Upstream and Downstream do the following book-keeping that will be used to see whether Eve has successfully cheated the game. For a link ℓ not adjacent to Eve, we let $\kappa_\ell(u)$ denote the *actual* fraction of exchanges during interval u where link ℓ was the farthest-upstream faulty link. Note that, because we model the congestion occurs at each link with probability *rho*, for link $\ell = (i, i+1)$ then the expected loss rate at link ℓ is $\mathbb{E}[\kappa_\ell(u)] = 1 - (1-\rho)^i$. (The actual value of $\kappa_\ell(u)$ will vary each time). We let $\kappa_{\mathsf{Eve}}(u)$ denote the fraction of exchanges during interval u where a fault occured on a link containing Eve (*i.e.* the fraction of exchanges for which there was no congestion and yet Bob fail recover the correct data for that exchange).

Game execution: the game runs as follows: $\mathsf{Init}(U_n)$ is used to create the shared randomness, and aux_i is given to Eve while the rest of the auxiliary information is given to Upstream and Downstream. Eve interacts with the oracles Source, Upstream, Downstream for a polynomial number of exchanges. At the

[12] One could generalize to the case where the probability of each link being down is different or may be dependent; our analysis extends straightforwardly to this case.

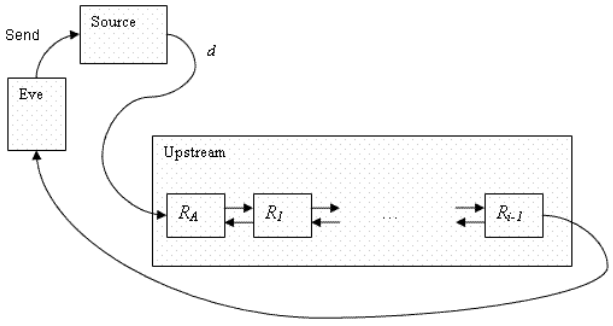

Figure 17: Eve's interaction with Source

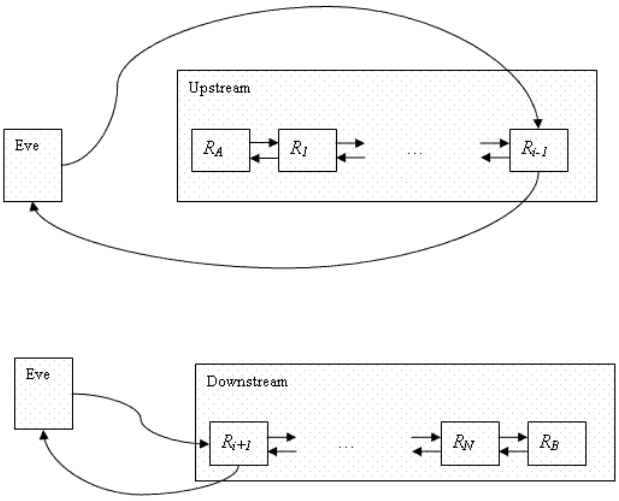

Figure 18: Eve's interaction with Upstream and Downstream

end of the game Alice runs FaultFunction to localize the faults. FaultFunction is different in the per-packet and statistical setting (we describe FaultFunction in the upcoming definitions).

We shall use this weak security game to prove our negative results. We now present our strong definition of security, that we use to prove the security of FL protocols.

Definition 4.6 (Unreliable-link strong-security game for FL)**.** In this game we give Eve more power. We allow Eve to occupy multiple nodes, adaptively choose her locations, and select the data that Alice transmits.

The game setting: This game runs in two phases and is more complex than the previous game. In the *training phase* a simulator oracle Simulator is given $\overrightarrow{\mathsf{aux}}$. Eve is given access to this oracle, which she can use by giving Simulator an input d, and Simulator will (honestly) simulate the entire network when Alice tries to send d to Bob. Simulator gives back to Eve the transcripts $\langle R_i, R_{i+1}\rangle(d)$ for every link in the network, as well as $\mathsf{FaultFunction}(\mathsf{aux}_A, \langle R_A, R_1\rangle(d))$ and $\mathsf{Decode}(\mathsf{aux}_B, \langle R_N, R_B\rangle(d))$. We call this the *record* of the exchange for d, and will denote it ω. Eve is allowed to interact with Simulator a polynomial number of times to produce polynomially many records $\omega_1, \ldots, \omega_{\ell_2(n)}$.

At the end of the training phase, Eve uses her records ω_i to decide on a subset $S \subseteq [N]$ to corrupt and enters into the *challenge phase*. In the challenge phase, Eve is given the auxiliary information of all the nodes in S. The rest of the auxiliary information is given to the oracles that Eve will be given access to. The oracles are defined in the following way. Divide $[N] \setminus S$ into blocks consisting of consecutive uncorrupted nodes. That is, let

$$B = \{(i,j) \mid i < j \text{ and } i,j \notin S, \text{ and } \forall k \in [i,j],\ k \notin S\} \tag{4.1}$$

define the blocks of good nodes, and let $b = |B|$ be the number of good blocks. We let $B_z = (i_z, j_z)$ denote the z'th block in B (in sorted order). Note that since Eve cannot corrupt Alice and Bob we always have $i_1 = A$ and $j_b = B$. Figure 19 illustrates this block structure. We will let each block of consecutive good nodes be simulated by an oracle; with a slight abuse of notation we let B_z denote the oracle associated with block B_z.

Oracle interactions: Eve is given access to oracles Source, $B_1, \ldots, B_b$. Now, in contrast to Definition 4.5, Eve is allowed to specify to Source the data packet she wants transmitted, which then sends it to B_1 which simulates Alice sending d to Bob, and returns the response of B_1 to Eve. Eve is subject to the constraint that she does not query a d twice, nor is she allowed to query any d that was previously queried in the training phase. Apart from these two differences, the picture looks the same as Figure 17.

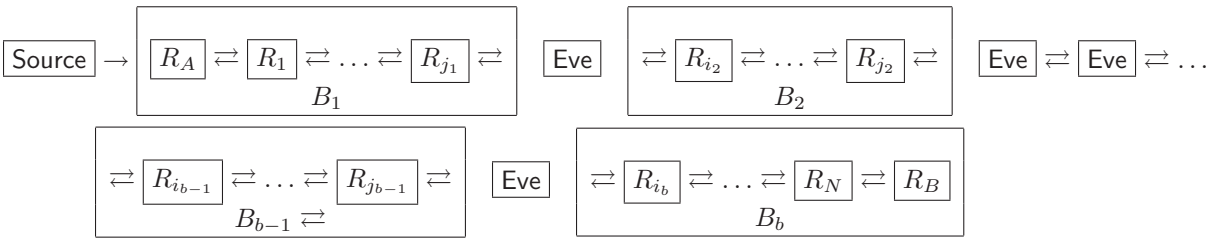

Figure 19: Block structure

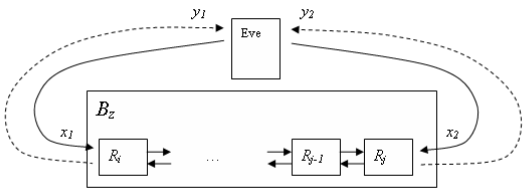

Figure 20: Interaction of Eve with oracle B_z.

Eve interacts with B_1, B_b in exactly the same way as she did with Upstream, Downstream in Definition 4.5. Eve interacts with B_z for $z \notin \{1, b\}$ by sending a pair of input packets (x_1, x_2) (where either of x_1, x_2 may be empty). The oracle B_z simulates $R_{i_z}, \ldots, R_{j_z}$ when R_{i_z} is given input x_1 coming from R_{i_z-1} and R_{j_z} is given input x_2 coming from R_{j_z+1}. It returns an output (y_1, y_2) which is the outputs of R_{i_z}, R_{j_z} respectively. This is illustrated in Figure 20.

Congestion: as with the weak security definition, each of the oracles simulates each link being down in each exchange with probability ρ.

Bookkeeping: During the challenge phase, during interval u, the fault rates $\kappa_\ell(u)$ for link ℓ not adjacent to S and $\kappa_{\mathsf{Eve}}(u)$ are recorded as in the weak security game of Definition 4.5.

Game execution: as with weak security, the game begins by the distribution of the auxiliary information $\overrightarrow{aux} = \mathsf{Init}(U_n)$. Then Eve gets to play the training phase, at the end of which she chooses her set of nodes to corrupt and receives the auxiliary information for those nodes. She then plays the challenge phase for at most a polynomial number of exchanges. She may use her past interactions, her knowledge aux_i for $i \in S$, as well as the records from the training phase ω_i in any (efficient) way she likes.

4.2.1 Per-packet Definitions

Again we begin with the per-packet definition. In per-packet security for fault localization, the output of $\mathsf{FaultFunction}(\cdot)$ is a vector of T entries (where T is the number of exchanges for which the FL game is played), where for each exchange, there is an entry is either a link ℓ that Alice decides is faulty, or the the value NoFault.

Definition 4.7 (Per-packet security for FL)**.** We say a fault localization scheme is weak (resp. strong) per-packet secure if no efficient adversary Eve playing the weak (resp. strong) FL game of Definition 4.5, (resp. Definition 4.6) can break the following correctness and security conditions with non-negligible probability:
Correctness condition: In the absence of an adversary, then for every exchange d the fault vector $\mathsf{FaultFunction}(d)$ is $\perp$ if no congestion occurred and $\mathsf{FaultFunction}(d) = \ell$ if the upstream-most congested node was ℓ.
Security condition: Here, we keep track of a list $\mathcal{L}_{\mathsf{Fault}}$ which has T entries, one for each exchange.

Each entry is either:

1. The symbol $\perp$ if the packet reached Bob successfully.
2. The link ℓ where the packet was *actually* dropped if it was dropped due to congestion.
3. The symbol Eve if the packet was not dropped by congestion and did not reach Bob.

We say that the protocol is secure if,

- for every entry d in $\mathcal{L}_{\mathsf{Fault}}$ equal to Eve, then Alice outputs $\mathsf{FaultFunction}(d) = \ell'$ where $\ell' \in S'$ is a link adjacent to Eve, and
- for every entry d in $\mathcal{L}_{\mathsf{Fault}}$ equal to ℓ, then Alice outputs $\mathsf{FaultFunction}(d) = \ell'$ where either $\ell' = \ell$ (the link at which the congestion really occurred) or $\ell' \in S'$ (a link at adjacent to Eve).

That is, the protocol is secure if every time Eve drops a packet that is not dropped by congestion, then Alice implicates a link adjacent to Eve, and that Eve cannot cause Alice to localize a fault to an innocent (not-congested) link.

4.2.2 Statistical Definitions

In statistical security, we divide the duration of the game in intervals, (where each interval consists of T_0 exchanges, and each exchange is all the communication associated with sending a single packet). Here, the output of $\mathsf{FaultFunction}(\cdot)$ is a vector of length equal to the number of interval occurring in the FL game, where each entry corresponds to a single *interval*. Each entry is either the value $\mathsf{NoFault}$ if Alice decides not to raise any alarms during that interval, or a set of links ℓ for which Alice decides to raise an alarm.

Definition 4.8 ($(\vec{\alpha}, \beta, \delta, T_0)$-statistical security for FL)**.** Here, $\vec{\alpha}$ is a vector of fault rates, one for each link, which are false alarm thresholds for each link. β is a detection threshold above which the path is faulty. We require that $\sum_\ell \alpha_\ell < \beta$. T_0 is the number of exchanges during each interval (and is typically a function of $\vec{\alpha}, \beta, \delta$). We say a fault localization scheme is weak (resp. strong) $(\vec{\alpha}, \beta, \delta, T_0)$-statistically secure if no efficient adversary Eve playing the weak (resp. strong) FL game of Definition 4.5, (resp. Definition 4.6) can break the following correctness and security conditions:
Correctness condition: We say that an FL protocol is correct if, for any interval u and conditioned on all previous intervals, then: for each honest link ℓ (that is not adjacent to a node occupied by Eve), the probability that the fault rate for link ℓ is below the false alarm threshold $\kappa_\ell(u) \leq \alpha_\ell$ but Alice raises an alarm for link ℓ is bounded by $\frac{\delta}{N} + \varepsilon(n)$.
Security condition: We say that an FL protocol is secure if, for any interval u and conditioned on all previous intervals, then: the probability that the fault rate $\kappa_{\mathsf{Eve}}(u)$ exceeds the detection threshold, and Alice outputs either $\mathsf{NoFault}$ or a link $\ell \notin S$ not adjacent to Eve, is bounded by $\delta + \varepsilon(n)$.

4.2.3 Properties of Security Definitions

We briefly comment on some properties of our security definitions. (We refer the reader back to Section 3.2.3 for more comments.)

Congestion: Since FL localizes faults to particular links, we want to accurately attribute faults, even ones caused by normal congestion, to the links on which they occurred. In order to do this, we need to explicitly model the congestion on every link, even those not adjacent to Eve. As such, we incorporate the probabilistic model of congestion (using uniform probability ρ for congestion on each link) into the game setting. Note that any probability distribution for congestion-related faults can be used here; we chose to use a uniform distribution for simplicity. The important thing to note is that congestion-related faults occur with non-negligible probability and that Eve does not control the occurrence of such faults.

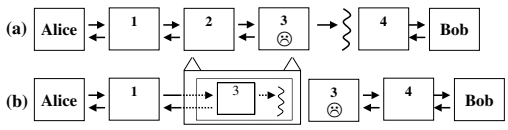

Figure 21: Necessity of keys and crypto.

Monotonicity: In this case, our security definition does *not* imply monotonicity (as defined in Section Section 3.2.3). We discuss how our protocols can be made monotone by including additional MACs or signatures in this technical report.

4.3 Fault Localization: Negative Results

The security guarantees provided in FL are strictly stronger than those provided in FD; it follows that FL protocols require strictly more overhead than FD protocols. Here we show that, even in the setting of source routing with symmetric paths, in any secure per-packet FL scheme (1) Alice requires shared keys with Bob and the intermediate nodes, (2) Alice, Bob and each intermediate node must perform cryptographic operations, and (3) Alice and each intermediate node must modify storage for each packet sent. Again, our results on the necessity of keys and cryptography rely on the assumption that Eve actively forges acks, while our result on the necessity of storage holds even when Eve can only selectively drop packets.

4.3.1 Keys are necessary at every node

We now show that in secure fault localization, each node must have some secret information.

Proposition 4.9. *For any fault localization scheme, for any $i = 2, ..., N, B$ if aux_j and aux_i are independent for some $j < i - 2$, or if it is easy to guess aux_i, then the scheme is not secure (in any sense!).*

Proof. We prove this in the contrapositive. Suppose that there exists a router i who shares no secrets with any $j \leq i - 2$. Then consider an adversary Eve who controls node $i - 1$. In Figure 21, we set $i = 3$ and represent node 3's sad lack of secrets with a frowney. In case (a) of Figure 21, node 4 is unreachable so that by robustness (see Section 4.2) Alice must localize the fault to link $(3, 4)$. However, because neither Alice nor node 1 share any secrets with node 3, Eve knows everything that Alice and node 1 know about node i. It follows that in case (b) Eve can *simulate* router 3 in a way that convinces $A, 1$ that 4 was indeed unreachable. Because case (a) and (b) are indistinguishable for routers $A, 1$, in case (b) Alice will localize the fault to link $(3, 4)$ that does not contain Eve. It follows that FL protocol is insecure if node i has no secret information. ■

For the remainder of the paper we will assume that there is indeed shared secret information between *Alice* and each intermediate node (not just between each intermediate node and some upstream node at least two hops away). To see why we do this, suppose that an adversary Eve who controls all the nodes $1, \ldots, i - 1$, between node i and Alice. In this setting, the scheme can only be secure if node i shares secret keys with *Alice*. As such, we will assume that for each node i, Alice and R_i either have shared secret keys or a public/private key pair.

4.3.2 Cryptography is necessary at every node

We can also prove that the keys at each node must be used "in a cryptographically-strong manner". At a very high level, the structure of this proof resembles the structure of our previous proof; refer again to Figure 21, where this time $i = 3$'s frowney represents the sad fact that R_i does not perform cryptography. Again we will show that Eve controlling R_2 can break security by simulating R_3 in a

manner that convinces Alice that link $(3, 4)$ failed. However, now R_3's key is kept secret from Eve, so Eve needs to find some way to *learn* R_3's key in order to simulate R_3. We note that case (a), where R_4 is unreachable for the duration of an exchange, will occur relatively frequently during the course of the security game due to *congestion*. Because node R_3 does not use cryptography, each time case (a) occurs Eve at node R_2 *observes* the way R_3 reacts to congestion, and uses her observations *learn* 'a part of node R_3's secret key'. After seeing sufficiently many examples of case (a), Eve has learned enough to be able to simulate R_3's reaction to congestion and break security as in case (b).

In the rest of this section, we formalize this intuition (Theorem 4.12 for the per-packet setting, and Theorem 4.15 for the statistical setting). The algorithm used to learn the key is a variant of the learning algorithm of [22] which we discuss in Appendix A, and the proof is an oracle separation in the blackbox model of Impagliazzo and Rudich [17]. .

The black-box model: Cryptographic reductions usually consist of two distinct parts: the construction and the proof of security. The construction shows how to use one primitive (such as a PRF) in order to build another secure system (for example encryption or FD). The proof of security shows how to convert an efficient adversary for the secure system (e.g. encryption or FD) into an efficient adversary for the original primitive (e.g. a PRF). When we say that a reduction is *fully black box* we mean that both the construction and the proof of security are black box: the construction uses the primitive in a black-box manner and the proof of security uses the adversary for the secure system in a black-box manner to construct the adversary for the original primitive. Following [27], when we say *semi-black box* we mean that only the construction is black-box, and the proof of security may be arbitrary. For example, the proof of security may take the code for the efficient adversary and only run specific sub-routines. All of our negative results apply to reductions that are semi-black box.

Technical overview of our proof: In this section we show that, for any fault localization scheme that is unreliable-link per-packet weakly secure, and whose construction is *fully black-box*, it follows that each intermediate node must performs some cryptography. This negative result does *not* rule out non-blackbox constructions where intermediate nodes do not perform cryptography; however all known practical cryptographic constructions are blackbox, so for all practical purposes, this does not weaken our results. To prove our result, *we will exhibit an oracle relative to which any fully black-box fault localization scheme where one node does not use cryptography cannot be unreliable-link weakly secure.* Since any fully black-box construction that is secure must be secure relative to all oracles, this suffices to prove that such constructions are unconditionally insecure. This methodology follows that of [17]. The proof uses the argument of Figure 21, where node i does not preform cryptographic operations (*i.e.* does not call the ideal cipher IC), and Eve at node $i-1$ implicates an innocent link $(i, i+1)$, this time by *using the power of the oracle.*

Our results are even more general; they can be extended to show that to prove the security of a *semi*-black-box construction of FL, one must in the process prove that $\mathbf{P} \neq \mathbf{NP}$. We do not elucidate this result further since the extension is essentially identical to that of [17] and we refer the reader to that work.

PSPACE, Ideal-Cipher Oracle: Our oracle will be a $(\mathbf{PSPACE}, \mathsf{IC})$ oracle. That is, the oracle consists of **PSPACE**, an oracle that solves **PSPACE**-complete problems, and IC an ideal cipher oracle [12]. Informally, we can think of the ideal cipher (IC) as an idealization of any symmetric cryptographic primitive. For example, the IC can model 'perfect encryption', an encryption scheme that cannot be broken even by a computationally unlimited adversary. More formally, the IC models a truly random permutation. A truly random permutation is just like a truly random function, where every input is mapped to a random output (as described in Definition 3.17) except that now the permutation is one-to-one: every every output maps to a unique input, so that (unlike a truly random function) the truly random permutation is *invertible*. The formal definition of an ideal cipher oracle follows:

Definition 4.10. An *ideal cipher oracle* is an oracle that gives access to a family of truly random permutations. That is IC takes as input a key k, and a direction, forward or reverse f, r. Then, $\mathsf{IC}_k(\mathsf{f}, \cdot)$, is a truly random permutation, and $\mathsf{IC}_k(\mathsf{r}, \cdot)$ is its inverse, i.e. $\mathsf{IC}_k^{-1}(\mathsf{f}, \cdot) = \mathsf{IC}_k(\mathsf{r}, \cdot)$.

Thus, when we say that "a node uses cryptography", we really mean that a node calls the IC during its

execution. We chose the IC to model the "use of cryptography" in our proof because it is invertible and known to be at least as strong as the random oracle model. In our proof, all nodes (even the ones that do not use cryptography) have access to a PSPACE oracle.

PSPACE is the set of decision problems that can be solved by a deterministic or nondeterministic Turing machine using a polynomial amount of memory and unlimited time [**?**]. Intuitively, an oracle that solves PSPACE-complete problem can be used to solve 'computationally inefficient' decision problems *e.g.* NP-complete problems.

In our setting, we can intuitively think of the PSPACE-oracle as an oracle that can invert any function (or solve any problem) that was *not* generated by the IC oracle. That is, since we can use the **PSPACE** oracle to invert any functions that do not come from IC, relative to the oracle $(\mathbf{PSPACE}, \mathsf{IC})$ the only cryptographic hardness that exists in the setting of our proof comes from the IC. This intuition was formalized in the result of [18].

Theorem 4.11 (Jerrum Valiant Vazirani 1986 [18])**.** *Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a function that is computed by a (possibly randomized) algorithm with access to a* **PSPACE** *oracle, $M^{\mathbf{PSPACE}}$ (here $\ell(n)$ is polynomial). Then there exists another (possibly randomized) algorithm with access to a* **PSPACE** *oracle, $N^{\mathbf{PSPACE}}$ that, for every $z = f(x)$, with probability $\geq 1/2$ outputs a uniform element of $\{y \mid f(y) = z\}$.*

We interpret this as follows. We consider a world in which all nodes have access to a PSPACE oracle. In this world, the result of JVV [18] implies that *no function is one-way* (*i.e.* hard to invert) except the IC.

Why fully black-box reductions must be secure relative to $(\mathbf{PSPACE}, \mathsf{IC})$: One can see now why any fully black-box reduction that is not secure with a $(\mathbf{PSPACE}, \mathsf{IC})$ oracle cannot be secure: it is not hard to believe (and can be shown by similar reasoning to that of [17, **?**]) the IC oracle is secure against *any* machine that only queries IC a polynomial number of times. Our proof below will show an adversary relative to $(\mathbf{PSPACE}, \mathsf{IC})$ that beats any FL scheme where one node does not use cryptography (*i.e.* does not call IC), and that adversary will only query IC a polynomial number of times. By the security reduction of the FL scheme this gives a procedure to break IC using only a polynomial number of queries, which is a contradiction, and so the FL scheme cannot be secure.

4.3.2 The per-packet setting

Theorem 4.12. *Fix any fully black-box fault localization scheme. If any node R_i does not use cryptography, then the scheme is not unreliable-link per-packet weakly secure.*

Again our proof essentially uses the result of Naor-Rothblum [22], who give an information-theoretic (inefficient) algorithm for (δ, ε)-learning adaptively-changing distributions (ACD). We describe this result in Appendix A. Recall that [22]'s learning algorithm makes use of a algorithm that inverts i.o. one-way functions, and is therefore inefficient. However, in our setting all parties have access to the **PSPACE**-oracle, that by Theorem 4.11 allows them to invert any one-way function (apart from those computed by the ideal cipher IC). Therefore, it follows that *[22]'s algorithm becomes efficient relative to a* **PSPACE** *oracle.* More formally, it follows from Naor-Rothblum's result [22] that:

Theorem 4.13 (Naor-Rothblum [22])**.** *There exists an efficient algorithm relative to a* **PSPACE** *oracle*[13] *$L^{\mathbf{PSPACE}}$ that (δ, ε)-learns any ACD.*

We will show that this algorithm $L^{\mathbf{PSPACE}}$ can be used to construct an adversary for any fault localization scheme where a node does not use cryptography.

Proof of Theorem 4.12. Let R_i be the node that does not use cryptography, *i.e.* it never calls IC. We will show an adversary Eve that sits at node $i-1$ and breaks the unreliable-link per-packet weak security of the scheme.

[13] When we say that the algorithm is efficient relative to a PSPACE oracle, we say that the algorithm itself runs in polynomial times and it makes calls to a PSPACE oracle.

The reduction proceeds as follows. Eve uses the algorithm of [22] (which is efficient given the **PSPACE** oracle) to learn the ACD given by transcripts of the FL scheme. She uses this to construct a candidate secret initial state x' and corresponding auxiliary information $\overrightarrow{\mathsf{aux}} = \mathsf{Init}(x')$. She then impersonates R_i using this candidate state x', and break security as in Figure 21.

The proof that doing this breaks security has two main steps: first we apply the result of [22] to say that the candidate state x' behaves "almost like" the true secret state x, even if we condition the distribution on link $(i, i+1)$ being congested. Then we apply Lemma A.3 to show that, with x', Eve can successfully simulate transcript for which link $(i, i+1)$ is congested. That is, we will show the candidate state x' can be used to simulate transcripts for which the link $(i, i+1)$ was congested, so that Eve can cause Alice to implicate an innocent link $(i, i+1)$ and break per-packet security.

Notation: The transcripts of the FL scheme are given by the ACD (Init, D), where D is a sample of the transcript at $\langle R_{i-1}, R_i \rangle$ when all the nodes are honest, and given random data d and using the ideal cipher IC. We identify the IC with a random string that describes all the calls to the ideal cipher, which is of polynomial length since the game runs for at most polynomial number of steps. The randomness of the IC is stored along with the randomness used by the R_j $(j = 0, \ldots, N+1)$, and so when computing subsequent transcripts we are sure that the responses to queries to IC are consistent.

We let D_H^x be transcript on the link $(i-1, i)$ for a uniformly random data packet given secret auxiliary information $\overrightarrow{\mathsf{aux}} = \mathsf{Init}(x)$, history $H = (\mathsf{IC}, \vec{W}, \vec{\tau})$ where the IC contains the ideal cipher, $\vec{W}$ contains all the randomness used by the all the nodes $R_1, \ldots, R_{i-1}$, and $\vec{\tau}$ is the transcripts of the previous segmnets.

Output x' "behaves like" x: Using the algorithm of [22], we have an algorithm $L^{\mathbf{PSPACE}}$ that sees $\mathrm{poly}(n)$ samples from the distribution and is able to generate a secret initial state x' such that with high probability, a sample transcript generated according to x' (namely *all* the R_j use the auxiliary information obtained from $\overrightarrow{\mathsf{aux}'} = \mathsf{Init}(x')$) and that is consistent with previous transcripts $\overrightarrow{\tau}$ is ε-close to a transcript generated using the true secret auxiliary information information x.

Notice here that $L^{\mathbf{PSPACE}}$ will treat the randomness of the ideal cipher the same as shared secret randomness between the nodes. That is, $L^{\mathbf{PSPACE}}$ does not try to learn the true queries and answers that the R_j made to IC, but instead "makes up" its own queries and answers that are consistent with the transcripts it sees. This would be problematic if in the end it were trying to imitate a node that also called the IC since the R_j could then distinguish between the true answer of the IC and fake answers, but it is not a problem for our purpose because we are trying to imitate a node R_i that does *not* call the IC.

Also notice that during the learning process, the IC oracle is called only a polynomial number of time, namely it is called whenever the honest R_j query it in order to produce honest transcripts. The IC oracle is never called by $L^{\mathbf{PSPACE}}$ for its own computations.

Unfortunately we are not done, because in this setting Eve is not interested sampling from D_H^x, but rather from $(D_H^x)^{R_{i+1}=\perp}$ *i.e.* the distribution of the next exchange *conditioned* on link $(i, i+1)$ being the upstream-most link that is down. (She needs to do this in order to impersonate node i when link $(i, i+1)$ is down as in Figure 21). Fortunately, due to *congestion* this happens with constant probability $\rho' = \rho(1-\rho)^i$. We will show (later) that once we are able to sample from $D_H^{x'}$ (without any conditioning) then, our samples conditioned on $(i, i+1)$ being down are still close to the true D_H^x.

Claim 4.14. *Suppose $L^{\mathbf{PSPACE}}$ outputs a candidate secret x' such that sampling from $D_H^{x'}$ is ε-close to D_H^x. Then the distributions conditioned on link $(i, i+1) = \perp$, namely $(D_H^{x'})^{R_{i+1}=\perp}$ and $(D_H^x)^{R_{i+1}=\perp}$ are ε/ρ'-close.*

So we know that with probability ρ' the sample $D_H^{x'}$ generated by Eve will have $(i, i+1)$ being congested, and conditioned on this it is ε/ρ'-close to the true distribution $(D_H^x)^{R_{i+1}=\perp}$ when $(i, i+1)$ is congested.

Using x' allows Eve to cheat: Now we can apply Lemma A.3. Suppose that Eve ran $T-1$ exchanges to learn x', and she wants to cheat on the T'th exchange. She will do so by simulating the T'th exchange using x' and pretending that $(i, i+1)$ is down. We want to calculate the probability

$$\Pr[Z'] = \Pr[\text{Alice localizes } T\text{'th exchange to } (i, i+1) \text{ with Eve}]$$

We use the shorthand 'with Eve' to represent the idea that Eve at node $i-1$ attempts to use her learning algorithm $L^{\mathbf{PSPACE}}$ to impersonate an exchange where link $(i, i+1)$ is down as in Figure 21. Using even shorter-hand, we call the event above Z'. We will try to calculate this probability by relating it to

$$\Pr[Z] = \Pr[\text{Alice localizes } T\text{'th exchange to } (i, i+1) \text{ without Eve} \mid (i, i+1) \text{ congested}] \approx 1$$

where the shorthand 'without Eve' means that node $i-1$ behaves honestly. We call this event Z. Here the approximate equality is up to an error negligible in n. The remainder of this proof shows that, because x' "behaves like" x, these two probabilities $\Pr[Z]$ and $\Pr[Z']$ are close. As such, it follows that Eve at node $i-1$ can drop a packet in the the T'th exchange and use her learning algorithm to successfully simulate an exchange when link $(i, i+1)$ was down, and thus break security.

Probabilities $\Pr[Z]$ *and* $\Pr[Z']$ *are close:* We start by applying Lemma A.3 in the following manner: We set

- A to be the routers $R_A, R_1, \ldots, R_{i-1}$ using the true IC and the true secret x
- B to be the honest router R_i using the true secret x, given the past transcripts $\vec{\tau}$, and conditioned on the link $(i, i+1)$ being congested
- A' to be the honest routers $R_A, R_1, \ldots, R_{i-1}$ using the candidate secret x' and using a fake IC' consistent with the past transcripts $\vec{\tau}$
- B' to be the simulation of R_i using the candidate secret x', given the past transcripts $\vec{\tau}$, and conditioned on $(i, i+1)$ being congested.

Now, the critical aspect of this proof is that we can apply Lemma A.3 because *we know that B is independent of A since R_i does not query the ideal cipher (because we assumed that R_i does use cryptography).* R_i's algorithm depends only on the secret x and B's own private randomness that he samples *independently* of the secrets and private randomnesses used by $R_A, R_1, ..., R_{i-1}$. Notice that if R_i did query the IC, then A and B would be dependent because of on their mutual use of the same IC oracle.

Let the randomness of A be $\mathsf{IC} \circ \vec{W}$, where IC is a random string describing the instantiation of the ideal cipher used by all the nodes (expect of course for poor R_i), and $\vec{W}$ is the internal randomness used by $R_A, R_1, \ldots, R_{i-1}$ and including the data d. Note that $\vec{W}$ and IC are for the current exchange *and* all the previous exchanges. Thus by Lemma A.3 we have that (the random variable containing) the 'impersonated R_i' using the auxiliary aux'_i learned by Eve, *i.e.* $R_i(\mathsf{aux}'_i)$, is $\ell\varepsilon/\rho'$-statistically close to (the random variable containing) the true R_i. That is[14]

$$\Delta(\ \mathsf{IC} \circ \vec{W} \circ \langle R_{i-1}, R_i(\mathsf{aux}'_i)\rangle^{R_{i+1}=\perp}\ ,\ \ \mathsf{IC} \circ \vec{W} \circ \langle R_{i-1}, R_i\rangle^{R_{i+1}=\perp}\ ) = \ell\varepsilon/\rho'$$

Next notice that impersonated transcript $\mathsf{IC} \circ \vec{W} \circ \langle R_{i-1}, R_i(\mathsf{aux}'_i)\rangle^{R_{i+1}=\perp}$ completely determines the conversation between Alice and R_1 given the impersonated $R_i(\mathsf{aux}'_i)$, *i.e.* $\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1\rangle^{R_i(\mathsf{aux}'_i),\ R_{i+1}=\perp}$. Likewise the honest transcript, $\mathsf{IC} \circ \vec{W} \circ \langle R_{i-1}, R_i\rangle^{R_{i+1}=\perp}$ completely determines $\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1\rangle^{R_{i+1}=\perp}$. Thus, it follows that

$$\Delta(\ \mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1\rangle^{R_i(\mathsf{aux}'_i),R_{i+1}=\perp}\ ,\ \ \mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1\rangle^{R_{i+1}=\perp}) \leq \ell\varepsilon/\rho'$$

[14] We recall our notation here:

- $\langle R_{i-1}, R_i\rangle$ is the conversation between R_i (using his real secret key aux_i) and R_{i-1} (using his real secret key aux_{i+1}) during the exchange of interest.
- $\langle R_{i-1}, R_i(\mathsf{aux}'_i)\rangle$ is the conversation between R_i (using the 'learned' secret key aux'_i) and R_{i-1} (using his real secret key aux_{i-1}) during the exchange of interest.
- $\langle R_{i-1}, R_i(\mathsf{aux}'_i)\rangle^{R_{i+1}=\perp}$ is the conversation between R_i (using the 'learned' secret key aux'_i) and R_{i-1} (using his real secret key aux_{i-1}) during the exchange of interest when node R_{i+1} becomes unreachable (*i.e.* because link $(i, i+1)$ goes down).

Notice that since $\mathsf{IC}, \vec{W}$, and the transcript between R_A, R_1 completely determines the outcome of applying the FaultFunction function to this particular exchange. Since by robustness

$$\Pr[\mathsf{FaultFunction}(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_{i+1}=\perp}) = (i, i+1)] = 1$$

we get that

$$\begin{aligned}|\Pr[\mathsf{FaultFunction}(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_i(\mathsf{aux}_i'),R_{i+1}=\perp}) = (i, i+1)] \\ - \Pr[\mathsf{FaultFunction}(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_{i+1}=\perp}) = (i, i+1)]| \leq \ell\varepsilon/\rho'\end{aligned}$$

But this means that

$$\begin{aligned}|\Pr[Z'] - \Pr[Z]| =& |\Pr[\mathsf{FaultFunction}(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_i(\mathsf{aux}_i'),R_{i+1}=\perp}) = (i, i+1)] - \\ & \Pr[\mathsf{FaultFunction}(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_{i+1}=\perp}) = (i, i+1)]| \\ \leq& \ell\varepsilon/\rho'\end{aligned}$$

This means that the probability that Eve successfully cheats when she samples from $D_H^{x'}$ is at least $1 - \ell\varepsilon/\rho'$, which breaks security since $\ell\varepsilon/\rho' \ll 1$. ■

Proof of Claim 4.14. Notice that in sampling both D_H^x and $D_H^{x'}$ the probability of the links being down are identical, *i.e.* they do not depend on the choice of x or x'. Let $Y_{\rho'}$ be a ρ'-biased coin flip.

Given a transcript D_H^x, there exists a (possibly inefficient) procedure to determine whether $(i, i+1)$ was the upstream-most congested link during the a single exchange.[15] Call this procedure down, so that $\mathsf{down}(D_H^x) = 1$ if $(i, i+1)$ was down in the exchange of interest, and it is 0 otherwise. Thus we have that statistical distance

$$\Delta(\mathsf{down}(D_H^x) \circ D_H^x,\ \ \mathsf{down}(D_H^{x'}) \circ D_H^{x'}) = \Delta(D_H^x, D_H^{x'}) \leq \varepsilon$$

Now, we say that $(D_H^x)^{R_{i+1}=Y_{\rho'}}$ is distributed as D_H^x if $Y_{\rho'} = 0$ and as $(D_H^x)^{R_{i+1}=\perp}$ if $Y_{\rho'} = 1$. Now, both $\mathsf{down}(D_H^x), \mathsf{down}(D_H^{x'})$ are distributed as $Y_{\rho'}$, since link $(i, i+1)$ is down due to congestion with probability ρ'. It follows that

$$\Delta(Y_{\rho'} \circ (D_H^x)^{R_{i+1}=Y_{\rho'}},\quad Y_{\rho'} \circ (D_H^{x'})^{R_{i+1}=Y_{\rho'}}) \leq \varepsilon$$

Using the properties of statistical distance, we know that for any random process F that depends on y then

$$\Delta(Y \circ F(Y), Y \circ F'(Y)) = \mathbb{E}_{y \in Y}[\Delta(y \cdot F(y), y \cdot F'(y))]$$

Using this property, and the fact that $\mathbb{E}[Y_{\rho'}] = \rho'$ we can conclude that

$$\Delta((D_H^x)^{R_{i+1}=\perp}, (D_H^{x'})^{R_{i+1}=\perp}) \leq \varepsilon/\rho'$$

which proves the claim. ■

4.3.2 The statistical setting

Our negative result for statistical security is considerably weaker. The main restriction will be that we require the distribution of each exchange to be independent and identical, regardless of the history of past exchanges. We call schemes that satisfy this restriction *per-exchange* schemes. However, note that Alice's fault function can look at the entire transcript of an interval when deciding whether that interval was faulty or not. For example, in a per-exchange schemes, Alice will never adaptively change the messages she sends to Bob based on observations of a high fault rate (*e.g.* a scheme where Alice turns on repetition coding only when she observes a high fault rate is not a per-exchange scheme.)

[15] We know that such a procedure exists because the set of transcripts where $(i, i+1)$ is the upstream-most congested link is (almost) disjoint from the set of transcripts where $(i, i+1)$ is not the upstream-most congested link because of robustness. They can overlap in a negligible fraction of transcripts, which we ignore here for simplicity's sake.

Theorem 4.15. *Fix any fully black-box* per-exchange *fault localization scheme. If any node R_i does not use cryptography, then the scheme is not unreliable-link per-packet $(\vec{\alpha}, \beta, \delta, T_0)$-statistically secure for any non-trivial settings of the parameters.*

Proof. This proof is very similar to the proof of Theorem 4.12, with a few changes. First, in the statistical setting, Eve breaks security by simulating R_i (conditioned on link $(i, i+1)$ begin down) for an entire *interval* causing Alice to (falsely) detect a fault rate of $\approx 100\%$ on link $(i, i+1)$. (This is in contrast to the per-packet case where Eve only simulates R_i for single exchange.)

Next, note that since the distribution of exchanges is independent of history (*i.e.* because we only consider 'per-exchange' schemes), we can omit the H from the notation and simply speak of the distribution D^x for a candidate secret x.

Next, from the same reasoning as in the proof of Theorem 4.12, we can use the modified [22] learning algorithm to produce a candidate secret x' such that sampling from $(D^{x'})^{R_{i+1}=\perp}$ is ε/ρ'-close to $(D^x)^{R_{i+1}=\perp}$. By the same reasoning as before, this means that

$$\Delta(\mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_i(\mathsf{aux}_i'), R_{i+1}=\perp}, \mathsf{IC} \circ \vec{W} \circ \langle R_A, R_1 \rangle^{R_{i+1}=\perp}) \leq \ell\varepsilon/\rho' \tag{4.2}$$

In the statistical setting, instead of having Eve impersonate the situation in Figure 21 for a single exchange as in the proof of Theorem 4.12, now we want Eve to impersonate the situation in Figure 21 for an *entire interval of T_0 exchanges.* What we will want to show is that

$$\Delta(\mathsf{IC} \circ \vec{W} \circ (\langle R_{i-1}, R_i(x') \rangle^{R_{i+1}=\perp})^{T_0}, \mathsf{IC} \circ \vec{W} \circ (\langle R_{i-1}, R_i \rangle^{R_{i+1}=\perp})^{T_0}) \leq T_0 \ell\varepsilon/\rho' \tag{4.3}$$

where $(\langle R_{i-1}, R_i \rangle^{R_{i+1}=\perp})^{T_0}$ is T_0 repetitions of the conversation between R_{i-1} and R_i during a single exchange with honest R_i, and R_{i+1} unreachable, and $(\langle R_{i-1}, R_i(x') \rangle^{R_{i+1}=\perp})^{T_0}$ is T_0 repetitions of the conversation between R_{i-1} and R_i during a single exchange with Eve simulating $R_i(x')$ using the 'learned' key x', and R_{i+1} unreachable.

We shall prove Inequality 4.3 using induction. To start the induction (setting $j' = 1$), we use Inequality 4.2. Next, we shall complete the induction by showing that if

$$\Delta(\mathsf{IC} \circ \vec{W} \circ (\langle R_{i-1}, R_i(x') \rangle^{R_{i+1}=\perp})^j, \mathsf{IC} \circ \vec{W} \circ (\langle R_{i-1}, R_i \rangle^{R_{i+1}=\perp})^j) \leq j \cdot \ell\varepsilon/\rho \tag{4.4}$$

holds for $j = j'$, then it also holds for $j = j' + 1$. We do this using the following claim, which we prove later:

Claim 4.16. *Let X be a random variable and let Y, Y', Z, Z' be random variables where, conditioned on a fixed value of X, then Y is independent of Y', Z' and Z is independent of Y', Z'. Then if $\Delta(X \circ Y, X \circ Z) \leq \varepsilon_1$ and $\Delta(X \circ Y', X \circ Z') \leq \varepsilon_2$, it holds that*

$$\Delta(X \circ Y \circ Y', X \circ Z \circ Z') \leq \varepsilon_1 + \varepsilon_2$$

This claim is the reason we restrict ourselves to per-exchange protocols: if the protocol had dependencies between the exchanges then we could not apply this claim. In particular, the distance may possibly blow up much faster than this claim guarantees.

Now, to apply Claim 4.16, we let

- $X = \mathsf{IC}$ where IC is the instantiation of the IC used by nodes $R_A, R_1, ..., R_{i-1}$.
- $Y = \vec{W}_{j'} \circ \vec{\tau}_{j'}$ is the transcript and internal randomness of $R_A, \ldots, R_{i-1}$ used in the first j' exchanges of the game with the honest router R_i (using the true secret x, conditioned on the link $(i, i+1)$ being congested).
- $Y' = W_{j'+1} \circ \tau_{j'+1}$ is the transcript and internal randomness used in exchange $j'+1$ of the game with the honest router R_i (using the true secret x, conditioned on the link $(i, i+1)$ being congested).

- $Z = \vec{W}_{j'} \circ \vec{\tau}_{j'}$ is the transcript and internal randomness used in the first j' exchanges of the game with the simulated router R_i using the *learned secret* x', conditioned on the link $(i, i+1)$ being congested.
- $Z' = W_{j'+1} \circ \tau_{j'+1}$ is the transcript and internal randomness used in exchange $j'+1$ of the game with the simulated router R_i using the *learned secret* x', conditioned on the link $(i, i+1)$ being congested.

Now, we argue that, conditioned on X, then

- Y and Y', Z' are independent because we only consider per-exchange FL protocols (where each exchange is independent and identically distributed). That is, what happens in exchange $j'+1$ is independent of what happened in the previous j' exchanges.
- Z and Y', Z' are independent for the same reason that Y and Y' are independent.

We now use this to apply Claim 4.16. Thus, from the above, we argue that, conditioned on X, then Y, Y', Z, Z' are all independent. Furthermore, by the inductive hypothesis we have $\Delta(X \circ Y, X \circ Z) \leq j'\ell\varepsilon/\rho'$. Finally, by Inequality 4.2 we have that $\Delta(X \circ Y', X \circ Z') \leq \ell\varepsilon/\rho'$. Thus, applying Claim 4.16 implies that Inequality 4.4 holds for $j = j' + 1$.

Completing the induction gives us Inequality 4.3.

We now complete the proof, following a similar argument as in the proof of Theorem 4.12. Notice in the honest case when R_{i+1} becomes unreachable for the duration of an interval of T_0 exchanges, *i.e.* $(\langle R_A, R_1 \rangle^{R_{i+1}=\perp})^{T_0}$, then Alice will implicate the link $(i, i+1)$ as having a fault rate of $\approx 100\%$ for this interval. Now, Inequality 4.3 means that in the case where Eve impersonates R_i, $(\langle R_A, R_1 \rangle^{R_i(\mathsf{aux}'_i), R_{i+1}=\perp})^{T_0}$, Alice will also implicate link $(i, i+1)$ with probability $\geq 1 - T_0\ell\varepsilon/\rho'$. Thus, if we set $\varepsilon < \frac{\rho'}{2T_0\ell}$, we have constructed an adversary sitting at node $i-1$ that forces Alice to implicate link $(i, i+1)$ with probability $> 1/2$ and so the scheme is not secure. ■

Proof of Claim 4.16. The derivation is proceeds as follows. From the properties of statistical distance, we know that

$$\Delta(X \circ Y \circ Y', X \circ Z \circ Z') = \mathbb{E}_{x \xleftarrow{R} X} \Delta(Y \circ Y' \mid x, Z \circ Z' \mid x)$$

Now, using the fact that by the triangle inequality, for A independent of B, D and C independent of B, D it holds that $\Delta(A \circ B, C \circ D) \leq \Delta(A \circ C) + \Delta(B \circ D)$, we get:

$$\begin{aligned} &= \mathbb{E}_{x \xleftarrow{R} X}[\Delta(Y \mid x, Z \mid x) + \Delta(Y' \mid x, Z' \mid x)] \\ &= \Delta(X \circ Y, X \circ Z) + \Delta(X \circ Y', X \circ Z') \\ &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

■

Remark 4.17. Note that in our argument for the statistical setting, we presented an Eve at node $i-1$ that breaks security by causing Alice to implicate link $(i, i+1)$ as having a fault rate of about 100%. In fact, it suffices for Eve to break $(\alpha, \beta, \delta, T_0)$-security by having Alice implicate link $(i, i+1)$ as having a fault rate of about β. The parameters used in the proof of Theorem 4.15 could be improved by having Eve simulate a fault a link $(i, i+1)$ for only βT_0 exchanges (rather than for T_0 exchanges). This remark also applies to our argument for the necessity of cryptography in statistical fault detection, see Section 3.3.

4.3.3 Modifying storage is necessary at each node

In this section, we prove that in secure FL, each node must modify storage for (almost) each packet that it monitors. We start with a proof for the per-packet FL, and then extend the proof to the statistical setting.

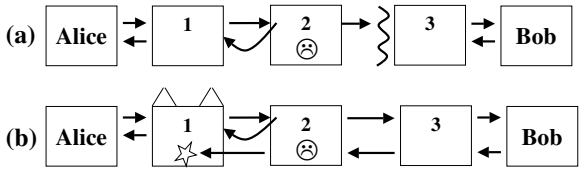

Figure 22: Necessity of storage.

Theorem 4.18. *In a per-packet weakly secure FL protocol, each intermediate node R_i for $i = 2, ..., N$ must modify storage at least once during each exchange.*

Proof. We prove this in the contrapositive using a timing attack. Assume (for the sake of the contrapositive argument) that we had an FL scheme, for which there was a single exchange during which node R_i does not modify storage. It follows that each time node R_i receives a message m about packet d, he must *immediately compute* and send his response(s). As such, we can categorize the messages that that node R_i sends as follows:

1. A message is sent in response to a message m sent by upstream neighbour node R_{i-1}. We say that the 'instigator', of the message is R_{i-1}. Let $m' = R_i(i-1, m)$ represent message sent by R_i in response to R_{i-1}'s message m.
2. A message is sent in response to a message m sent by downstream neighbour node R_{i+1}. We say that the 'instigator' of the message is R_{i+1}. Let $m' = R_i(i+1, m)$ represent message sent by R_i in response to R_{i+1}'s message m.
3. A message that is *not* sent in response to any message sent by the upstream or downstream neighbours of R_i. In this case, the instigator of the message is R_i himself. Let $m' = R_i(i, \perp)$ represent a message sent by R_i that R_i himself instigates.

Notice that this categorization gives information about the *timing* of each message. Therefore, node R_{i-1} can determine, for each message he receives from node R_i, whether it is of the first type or whether it is of the second or third type. Also, note that when R_i does not use storage it never generates messages of the third type.

As an overview of attack, refer now to Figure 22 and where $i = 2$. Node R_2's sad lack of storage means that upon receipt of a message corresponding to packet d from R_1, he must immediately transmit a response to downstream node R_3 and he may also need to immediately send a response back upstream to node R_1 (we represent this by the arc in Figure 22). In case (a) R_2 is unreachable. Notice that since R_2 does not access storage, R_2 transmits no further messages corresponding to packet d because *he forgot that he is waiting for a response from* R_3! In case (a) Alice localizes the fault to link $(2, 3)$. Now consider case (b), where Eve at R_1 forwards R_2's immediate response to her messages (again represented by the arc) but drops any communication that is not an immediate response to her message. From Alice's point of view, case (b) will look exactly like case (a), so Alice will implicate innocent link $(2, 3)$, and the FL protocol cannot be secure.

More formally, let $\tau = (R_i(\phi_1, m_1), R_i(\phi_2, m_2), ...)$ be the transcript of messages sent by R_i upstream to R_{i-1} during the exchange where R_i keeps no storage, where $\phi_j \in \{i-1, i, i+1\}$ is the instigator of the message. In case (a), when R_{i+1} is unreachable the transcript for case (a), which we call $\tau_{(a)}$, will contain no messages $R_i(i+1, \cdot)$ instigated by R_{i+1}, nor any messages $R_i(i, \perp)$ instigated by itself. Then, in case (b), if Eve controlling R_i using timing information to drop all messages instigated by R_{i+1} or R_i, then the transcript $\tau_{(b)}$ will be indistinguishable from $\tau_{(a)}$. It follows that Alice cannot distinguish between case (a) and case (b). But since Alice must implicate link $(i, i+1)$ because of robustness, it follows that she will also implicate $(i, i+1)$ in case (b), which breaks security. ■

Remark 4.19 (Border Cases)**.** The attack described above only applies to nodes with possibly malicious neighbours downstream of themselves, and nodes that may possibly become unreachable upstream of themselves. Since Bob has no downstream neighbour, and R_1 has no malicious upstream neighbour (since we assume Alice is never malicious), it is possible that there exists a secure FL protocol in which R_1 and Bob are exempt from the storage requirement.

We now extend the proof to the statistical setting, for *FL protocols based on sampling*. In an FL protocol based on sampling, each R_i decides if each packet it sends requires acknowledgement from Bob or not. We say that a packet that R_i requires Bob to acknowledge is *sampled*. Each node computes an estimated fault rate F_i which is the fraction of sampled packets that were not correctly acked. These estimates are then collected by Alice and used to compute link fault rates $F_\ell = F_i - F_{i+1}$ for each link $\ell = (i, i+1)$.[16]

Theorem 4.20. *Fix a $(\vec{\alpha}, \beta, \delta, T_0)$-weakly statistically secure FL protocol based on sampling. Let α_ℓ be the ℓ'th entry of $\vec{\alpha}$. For each intermediate node R_i for $i = 1, ..., N-1$, it must be that R_i modifies storage for at least $\Omega(\frac{1}{\beta} \ln \frac{1}{\frac{1}{2}+2\delta})$ exchanges.*

Proof sketch. The technical details of this proof are similar to those of Theorem 3.11 so we omit them and sketch the outline of the proof.

Suppose R_i modifies storage for T exchanges, where T is less than $\Omega(\frac{1}{\beta} \ln \frac{1}{1/2+2\delta})$-fraction of the exchanges that he samples. In particular this means that R_i samples at most T exchanges.

Consider the adversary that sits at R_{i+1} and randomly drops a β fraction of the packets. The probability that she gets detected by Alice is at most the sum of the following probabilities:

1. With probability at most $1/2$ Eve drops less than a β fraction of packets.
2. Then, since R_i only samples T packets with probability, his estimate F_i of the faultiness is non-zero with probability at most $(1-\beta)^T > \frac{1}{2} - 2\delta$.
3. R_{i+1} outputs an honest estimate F_{i+1}. Since the protocol is correct, the probability that the honest estimate $F_{(i+1,i+2)}$ is implicated is bounded by $\delta/N + \varepsilon(n)$.

Therefore the probability either link $(i, i+1), (i+1, i+2)$ is implicated is less than $1 - \delta$ even though more than a β fraction was dropped by R_i, which breaks security. ■

The result can be strengthened in the case of strong statistical security.

Theorem 4.21. *Fix a $(\vec{\alpha}, \beta, \delta, T_0)$-strongly statistically secure FL protocol based on sampling. Let $\alpha = \sum_\ell \alpha_\ell$. For each intermediate node R_i for $i = 2, ..., N$, letting $\eta_i = \frac{\beta}{\alpha} \sum_{j=i}^{N} \alpha_{(j,j+1)}$, it must be that R_i modifies storage for at least $\Omega(\frac{1}{\eta_i} \ln \frac{1}{\frac{1}{2}+2\delta})$ exchanges.*

The proof is essentially the same as for Theorem 4.20, except now Eve may occupy *every node except the one node R_i that does not use enough storage*. In this case, she will drop a β_ℓ fraction of packets along each link ℓ. As long as R_i outputs a bad estimate, Eve can fudge the other nodes' estimates to pretend that nothing bad happened. This is why we can replace β by η_i in the expression for the amount of storage.

4.4 Per-packet FL: Optimistic Protocol

Here we sketch our *optimistic per-packet FL protocol*, which can be thought of as a secure version of the Packet Obituaries [1] protocol. (Our protocol corrects a security vulnerability in [1]). This protocol

[16] We have failed to specify how Alice collects these estimates, but we assume that the collection method is secure. For our storage result we focus on the storage necessary during sampling and not during the final collection phase.

is instructive as an example of a secure per-packet FL protocol with little additional communication overhead beyond Per-Packet-Ack (with the addition of a few extra messages that are sent *only when a fault occurs.*), and because it demonstrates the use of *onion reports* (that appear again in our statistical FL protocol, Section 4.5).

In this section, we present our Optimistic Fault Localization Protocol, prove that it is strongly secure in the per-packet FL setting, then discuss the monotonicity of the protocol, and finally show how the protocol can be used in the setting of asymmetric paths.

4.4.1 Description of the Optimistic Protocol

Definition 4.22 (Optimistic Fault Localization Protocol)**.** Assume Alice shares a symmetric keys $[k_A^1\ k_A^2\ ...\ k_A^B]$ with all nodes on the path including Bob. The fault localization protocol consists of an (efficient) number of independent exchanges. In each exchange, algorithms $(R_A, R_1, ..., R_N, R_B)$ do the following:

The send phase: This identical to Per-Packet-Ack: Alice sends data packet d to Bob and Bob responds with ack $a = [B, h(d)]_{k_B}$. As before, Alice needs to store a data packet digest $z = h(d)$ and a time-out, but now we also require that *all intermediate nodes* R_i store a digest of the data packet $z_i = h(d)$ and ack $y_i = h(a)$ that they see. Each node (including Alice) keeps a hashtable that is indexed by hashing data packets d with h. As data packets d traverse the path downstream to Bob, each node i computes the packet digest z_i, marks row z_i in his packet hashtable, and sets a time-out period for z_i. Similarly, when the ack a traverses the path upstream to Alice, each node checks to see if the packet digest inside the ack $a = [B, h(d)]$ matches an marked entry in the node's packet hashtable. If it does, the node records $y_i = h(a)$ in the row of hashtable corresponding to $h(d)$. Notice that since packets may be tampered with as they traverse the data path, R_i may store a data packet digest z_i or ack digest y_i that is different from the data packet (or ack) sent by Alice (or Bob). When Alice receives an ack a, she checks if the ack a is authenticated with a valid MAC, and if the packet hash inside the ack a corresponded to a marked entry in her packet hashtable that has not timed out. If it does, Alice removes the corresponding marked entry from her packet hashtable.

The report phase: This phase is run if Alice detects a faulty exchange. (To detect faulty exchanges, Alice periodically checks her table to see if any entries have timed out.) The report phase on a faulty exchange begins as follows: Alice sends an *onion report request* to Bob, (A, z) to Bob, where $z = h(d)$ is the hash of the packet sent during the faulty exchange,[17] which tells all nodes along the path to prepare an onion report. To respond to request, each node i checks if z is marked in his packet hashtable; if it is, sets $\tau_i = (i, z, y_i)$ where y_i is the hash of the ack corresponding to z, otherwise he sets $\tau_i = (i, \perp, \perp)$ where $\perp$ is a special symbol denoting "missing". He then creates an onion report θ_i containing his information τ_i corresponding to the faulty exchange, which is concatenated to his downstream neighbor's onion report θ_{i+1} to form $\theta_i = [\tau_i, \theta_{i+1}]_{k_{i+1}}$. Notice that the onion reports are MAC'd in layers like an onion that cannot be split apart. [18] Also, when each node originally receives the onion request (A, z) from Alice, each node sets an "onion time-out" for exchange t which determines how long he should wait for his downstream neighbour to send their onion report. If the onion time-out expires, the node reports a missing onion report by setting $\theta_{i+1} = \perp$ and then proceeding to construct his own onion report as before. Alice uses the complete onion report θ_1 (that she gets from node 1) to localize the fault.

The onion report will consist of a bunch of nodes that are adjacent to Alice with *ABOVE* reports, and then, at some point one of the reports will become *BELOW*. The transition between *ABOVE* and *BELOW* reports will be implicated as the faulty link. More formally, the Fault function is computed by Alice as follows:

- If Bob's ACK $a = [B, h(d)]_B$ was valid, then no fault occurred (Alice outputs $\varnothing$).

[17] To prevent DoS attacks at intermediate nodes, it may be useful for (A, z) to be MAC'd by Alice.

[18] Onionizing reports prevents Eve from implicating a link between two innocent downstream nodes by selectively dropping reports. Selective dropping of reports and acks is a significant vulnerability of the FL protocols of [Awerbuch, SecTraceRoute].

- Output the most upstream link where the fault acknowledgements transition from ABOVE to BELOW.[19] If no such transition is found, output $(\perp, \perp)$. We classify onion reports θ_i as ABOVE or BELOW as follows:
  - A report $\theta_i = [i, \text{data-digest}, \text{ack-digest}, \theta_{i+1}]_i$ is ABOVE if
    - θ_{i+1} is present (we consider θ_{i+1} to be present when the first field of θ_{i+1} contains the name R_{i+1})[20] AND
    - The MAC on θ_i is valid AND
    - The data-digest field matches $h(d)$, the digest of the packet sent by Alice AND
    - the ack-digest field does *not* match $h([B, h(d)]_B)$, *i.e.* the hash of a valid ack for packet d.
  - Otherwise, a report is BELOW.

4.4.2 Security of the Optimistic Protocol

Proposition 4.23. *The Optimistic FL Protocol is per-packet strongly secure.*

Proof. We prove this in the contrapositive using a **three** step argument. First, we observe that there a number of ways in which Eve can win the security game.

1. Eve wins by dropping at least one packet but convincing Alice that no fault occurred. This is, $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| > 0$ but $\mathcal{L}_{\mathsf{Fault}} = \varnothing$. We call this event 'Eve evades detection.'
2. Eve wins by creating a fault but prevents Alice from localizing the fault. That is, $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$, but Eve prevents Alice from finding a transition between ABOVE fault acknowledgements and BELOW fault acknowledgement so that $(\perp, \perp) \in \mathcal{L}_{\mathsf{Fault}}$. (Note that we include in this case that event $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| = 0$ when Eve creates a false negative AND the event $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| > 0$ when Eve dropped a packet.) We call this event 'Eve prevents localization'.
3. Eve wins by implicates a link not containing herself. That is, $\exists (j, j+1) \in \mathcal{L}_{\mathsf{Fault}}$ such that $(j, j+1) \neq \varnothing$ and $(j, j+1) \notin S$. (Note that again we include in this case that event $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| = 0$ when Eve creates a false negative AND the event $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| > 0$ when Eve dropped a packet.) We call this event 'Eve implicates $(j, j+1) \notin S$'.

From the argument above we have that

$$\Pr[\text{Eve wins}] = \Pr[\text{Eve evades detection}] + \Pr[\text{Eve prevents localization}] + \Pr[\text{Eve implicates } (j, j+1) \notin S] \tag{4.5}$$

In subsequent Claim 4.25, Claim 4.26, and Claim 4.27, we provide bounds on each of the terms on the right side of the Equation 4.5 using contrapositive arguments. Our bound shall show that Pr[Eve wins] is negligible.

Claim 4.24. *For the Optimistic FL Protocol, for every* Eve *playing the FL game, it follows that*

$$\Pr[\text{Eve } \textit{evades detection}] \leq \epsilon_{mac}$$

where ϵ_{mac} is the MAC-advantage of the MAC Bob *uses to sign his ack.*

Proof. Suppose we had an Eve that wins by evading detection. Then, $|\mathcal{L}_{\mathsf{Sent}} \backslash \mathcal{L}_{\mathsf{Received}}| > 0$ but $\mathcal{L}_{\mathsf{Fault}} = \varnothing$. By definition of the Fault function it follows that there exists at least one exchange t for which the message (t, d) was not received by Bob, but a valid ACK $[B, h(d)]_B$ was received by Alice. It follows that there exists some node controlled by Eve (that is R_i for $i \in S$) that forged the ACK $[B, h(d)]_B$. Following

[19] For example, if after classifying the fault ACKs, Alice obtains $\theta_1 = ABOVE, \theta_2 = ABOVE, ..., \theta_i = ABOVE, \theta_{i+1} = BELOW, ..., \theta_N = BELOW$, localize the fault to link $(i, i+1)$.

[20] For example, θ_{i+1} is missing from θ_i when $\theta_i = [R_i, z_i, y_i, \theta_{i+2}]_i$

using a similar technique used to prove the security of per-packet-ACK, we can show that an Eve that evades detection can be used to construct an adversary Fred that forges MACs with non-negligible success probability. ■

Before we give bounds on the rest of the probabilities in Equation 4.5, we first need Claim 4.25.

Claim 4.25. *If $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$ it follows that there exists at least one exchange during which* Eve *tampered with the data message and/or with* Bob*'s ack message.*

Proof. From the definition of the protocol, $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$ only if during at least one exchange t of the protocol Alice received invalid acknowledgement $[B, h(d)]_B$. Recall that an ACK is valid if the MAC is valid AND $h(d)$ in the acknowledgement matches the data sent by Alice in exchange t. It follows that if an ACK is invalid, either Eve tampered with d sent downstream to Bob, or Eve tampered with the ACK $[B, h(d)]_B$ sent upstream back to Alice (so that the MAC on the ACK was invalid). ■

Claim 4.26. *For the Optimistic FL Protocol, for every* Eve *playing the per-packet strong FL game, it follows that*

$$\Pr[\mathsf{Eve}\ \textit{prevents localization}] \leq N\epsilon_{mac}$$

where ϵ_{otmac} is the MAC-advantage of the MACs used by the intermediate nodes to sign their onion reports.

Proof. Suppose we had an Eve that wins by preventing localization. That is $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$, but Eve prevents Alice from finding a transition between ABOVE fault acknowledgements and BELOW fault acknowledgement so that $(\perp, \perp) \in \mathcal{L}_{\mathsf{Fault}}$.

From Claim 4.25, since $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$ there exists at least one exchange during which Eve tampered either with the data message sent downstream by Alice or the ACK message sent upstream by Bob (or both). Lets assume, without loss of generality that *most upstream* position at which Eve tampered with the data message or ACK is position i for $i \in S$. It follows that if every node generated its onion report honestly, then $\theta_{i-1}, \theta_{i+1}$, the pair of onion reports generates by the up- and downstream neighbours of i, must have different entries in their data-digest fields, and/or different entries in their ack-digest fields. (Recall that an onion report has the form it $\theta_i = [i, \text{data-digest}, \text{ack-digest}, \theta_{i-1}]_i$. Recalling the definition of ABOVE and BELOW fault-ACKs, we claim that if every node generated its fault-ACK *honestly*, then there when $\mathcal{L}_{\mathsf{Fault}} \neq \varnothing$ there must be transition between ABOVE and BELOW fault acknowledgements.

Thus, if Alice does not see a transition between ABOVE fault-ACKs to BELOW fault-ACKs (*i.e.* $\mathcal{L}_{\mathsf{Fault}} = (\perp, \perp)$) it follows that Eve must have tampered with θ_{i+1} to change θ_{i+1} from a BELOW fault ACK to and ABOVE fault ACK. Furthermore, since ABOVE fault ACKs must have valid signatures, it follows that Eve must have forged the MAC of θ_{i+1}.

The formal reduction proceeds along these lines. In the reduction, Fred chooses $j \xleftarrow{R} [N]$, and then chooses keys all the MAC keys $k_A^1 ... k_A^N, k_A^B$, at random *except for* k_A^j and simulates the Simulator using the randomly chosen keys to compute all the MACs (on the onion reports and on Bob's acks) except for that of node R_j; for node R_i onion reports, Fred uses his signing oracle to compute the MAC.

Then, after Eve declares which nodes S she wants to corrupt, Fred continues his simulation using the randomly chosen MAC keys and his signing oracle until he finds a exchange during which he observes Eve tampering with either a data or an ACK message, but Alice fails to localize the fault. Let position i be the upstream-most position at which Eve tampered with a data or ACK message. When Eve generates onion report θ_i, Fred unpeels θ_i to get θ_{i+1} and outputs θ_{i+1} as his forged MAC. Since Eve prevent localization, she will have (changed θ_i from a BELOW fault ACK to an ABOVE fault ACK and also) forged the signature on θ_{i+1}. Therefore, conditioned on the event that $j = i$, it follows that Fred forges the MAC when Eve wins the fault localization protocol by preventing localization. That is

$$\epsilon_{mac} = \Pr[\mathsf{Fred}\ \text{forges}] = \Pr[\mathsf{Eve}\ \text{prevents localization} \cap\ \ i = j\ ] = \tfrac{1}{N} \Pr[\mathsf{Eve}\ \text{prevents localization}]$$

and the claim follows. ■

Claim 4.27. *For the Optimistic FL Protocol, for every* Eve *playing the per-packet strong FL game, it follows that*

$$\Pr[\mathsf{Eve} \text{ implicates } (j, j+1) \notin S] \leq N\epsilon_{mac}$$

where ϵ_{mac} is the MAC-advantage of the MACs used by the intermediate nodes to sign their onion report.

Proof. Suppose we had an Eve that wins in some exchange by implicating $(j, j+1) \notin S$. Now observe that if R_j and R_{j+1} are both honest and no congestion occurred during the exchange, it follows that neither of them ever tampers with the packet d sent by Alice or the ack message $a = [B, h(d)]_B$ sent by Bob. Since both R_j, R_{j+1} are honest, it follows that onion reports θ_j, θ_{j+1} will have identical information in their data-digest and ack-digest fields, and the MACs on θ_j, θ_{j+1} will be valid, and furthermore, θ_{j+1} will be properly 'onionized' or nested inside θ_j. Therefore, when R_j, R_{j+1} are both honest, it follows that $\theta_j, \theta_j + 1$ will either both be ABOVE or both be BELOW.

However, if Alice implicates link $(j, j+1)$, then Alice must see an ABOVE/BELOW transition between θ_j and θ_{j+1}. Therefore, we can conclude that if Alice implicates link $(j, j+1)$, then Eve must have intercepted and tampered with one or both of θ_j, θ_{j+1}. (For example, Eve could have removed θ_{j+1} from being nested in θ_j. Or, she could have modified the '$data'$ or 'ACK' fields in one or both of θ_j, θ_{j+1}. Or she could have tampered with the one-time-MAC in θ_j.) For Eve to have intercepted θ_j, θ_{j+1}, it follows that Eve occupies at least one node upstream of j.

Let $i \in S$ be the closet node upstream of j that is occupied by Eve. Observe that node $i+1$ is an honest node, and furthermore there are only honest nodes between i and j. Therefore, from the arguments above it follows that θ_{i+1} (which has θ_j, θ_{j+1} nested inside it) should *not* contain an ABOVE/BELOW transition at $j, j+1$. However, if $\mathcal{L}_{\mathsf{Fault}} = (j, j+1)$, it follows that Alice saw an ABOVE/BELOW transition at $(j, j+1)$. Therefore, we may conclude the Eve tampered with θ_{i+1} and forged $i+1$'s MAC.

The formal reduction is similar to the one in the previous claim. In the reduction, Fred again chooses $\ell \xleftarrow{R} [N]$, and then chooses all the MAC keys $k_A^1 ... k_A^N, k_A^B$, at random *except for* k_A^ℓ and runs the Simulator as before.

Then, after Eve says which positions S she wants to corrupt, Fred continues to simulate the FL game until he finds a exchange during which Eve implicates $(j, j+1)$, for i defined as above, he takes Eve's output θ_i, 'unpeels' it to get θ_{i+1} and outputs θ_{i+1}.[21] By the argument above, it follows that when Eve implicates $(j, j+1)$ it follows that she forged $i+1$'s mac. Now it follows that Fred wins when Eve implicates $(j, j+1) \notin S$ conditioned on the event that Fred places his signing oracle at the position that Eve forges a one-time-MAC. That is

$$\epsilon_{mac} = \Pr[\mathsf{Fred} \text{ forges}] > \tfrac{1}{N} \Pr[\mathsf{Eve} \text{ implicates } (j, j+1) \notin S]$$

and the claim follows. ∎

To complete the proof of Claim 4.23, we combine the bounds obtained in Claim 4.25, Claim 4.26 and Claim 4.27 with Equation 4.5 to find that

$$\Pr[\mathsf{Eve} \text{ wins}] \leq (2N+1)\epsilon_{mac}$$

and since ϵ_{mac} is negligible, it follows that Eve wins with negligible probability, and the Optimistic FL Protocol is secure. ∎

Remark 4.28 (Monotonicity)**.** As it stands the protocol is not monotone. To make it monotone,

- Each node R_i should only include an onion report θ_{i+1} in his own onion report θ_i when his upstream's neighbour onion report is validly signed.
- Each node should only store an ack in his packet hash table when the ack is validly signed by Bob.

[21] Observe that this argument also captures the situation in which Eve drops (or removes from the onion) some θ_j for $j \notin S$.

It is easy to make the FL protocol monotone in the *public-key* setting; simply give each intermediate node R_i the public key of Bob and upstream neighbour R_{i+1}, and have R_i discard onion reports θ_i and acks a that are invalidly signed.

In the *private-key* setting, making the protocol monotone is more challenging. One solution is

- Add pairwise keys k_i^{i+1} between adjacent nodes R_i and R_{i+1}, and have R_{i+1} add an extra MAC to his onion report (*i.e.* $\theta_{i+1} = [[...]_{k_A^{i+1}}]_{k_i^{i+1}}$ that R_i verifies and then strips off of θ_{i+1} before he includes it in his own onion report, θ_i.
- Have (in addition to the pairwise keys between Alice and intermediate nodes) pairwise keys between Bob and intermediate nodes $k_B^1, k_B^2, ..., k_B^N$. Bob could then construct his acks as follows

  $$[B : N, ...[B : 1, [B : A, h(d)]_{k_A^B}]_{k_B^1} ...]_{k_B^N}$$

  Each intermediate node would discard the ack if the MAC (intended for him) does not verify.

Such MAC'ing gymnastics (on Bob's acks) in the symmetric key setting could be avoided using the techniques employed in [2].

Another option is to have each node include all the acks he receives that correspond to a data packet d in his report (*i.e.* instead of including only 1 ack, he includes many acks). However, this is not good solution, it introduces the problem of adversaries that DoS an node by sending it many acks corresponding to a single packet so that it's memory overflows.

4.4.3 The Optimistic Protocol for Asymmetric Paths

Remark 4.29 (Asymmetric Paths)**.** The optimistic protocol can also be extended to the asymmetric path setting. We show how to do this here. Recall that in the asymmetric paths setting, the forward path (from Alice to Bob) is different from the reverse path (from Bob to Alice). We shall refer to N intermediate nodes on the forward path as $R_{f1}, ..., R_{fN}$ and M intermediate nodes on the reverse path as $R_{r1}, ..., R_{rM}$. The protocol proceeds as follows:

The send phase: This identical to Per-Packet-Ack: Alice sends data packet d to Bob and Bob responds with ack $a = [B, h(d)]_{k_B}$, except that this time, the N intermediate nodes on the forward path store the data-digest $z = h(d)$, and M intermediate nodes on the reverse path store the ack-digest $y = h(a)$. To do this, nodes on the forward path computes the packet digest z_i, marks row z_i in their packet hashtables, and sets a time-out period for z_i. Similarly, nodes on the reverse path compute $y = h(a)$ and store y in the row of hashtable corresponding to $h(d)$ (where $h(d)$ was the data-digest sent inside the ack $a = [B, h(d)]_B$).

The report phase: Again, this phase is run if Alice detects a faulty exchange. Again Alice wants to collect onion reports from the intermediate nodes on the forward and reverse paths. She does this as follows:

Alice sends an *onion report request* to node R_{f1}, (A, z) to Bob, where $z = h(d)$ is the hash of the packet sent during the faulty exchange. Then R_{f1} responds to the request by creating his onion report

$$\theta_{f1} = [f1, \text{data-digest}]_{k_A^{f1}}$$

and then forwards the onion report request (A, z) along with the onion response θ_{f1} downstream to node R_{f2}, who similarly creates onion report which is then forwarded along with the request downstream. The process continues until request (A, z) and onion report

$$\theta_{fN} = [fN, \text{data-digest}, \theta_{fN-1}]_{k_A^{fN}}$$

arrives at Bob. Bob then forwards the request and onion back upstream via the reverse path node R_{rM}, who creates an onion report

$$\theta_{rM} = [rM, \text{ack-digest}, \theta_{fN}]_{k_A^{rM}}$$

which it forwards, along with the onion report request upstream towards Alice. (Notice that the reverse path node is responding with an onion report the contains the ack-digest corresponding to z in onion request).

This process continues until Alice gets back the original report request and completed onion report θ_{r1}. The onion report θ_{r1} will consist of a bunch *BELOW* reports, followed by an *ABOVE* report. Again, the transition between *BELOW* and *ABOVE* reports will be implicated as the faulty link. More formally, the Fault function is computed by Alice as follows:

- If Bob's ACK $a = [B, h(d)]_B$ was valid, then no fault occurred (Alice outputs $\varnothing$).
- Output the first link in the onion report where the fault acknowledgements transition from ABOVE to BELOW.[22] If no such transition is found, output $(\perp, \perp)$. We classify onion reports θ_i as ABOVE or BELOW as follows:
  - A report from a node on the forward path $\theta_{fi} = [fi, \text{data-digest}, \theta_{fi-1}]_i$ is ABOVE if
    - θ_{fi-1} is present, AND
    - The MAC on θ_{fi} is valid AND
    - The data-digest field matches $h(d)$, the digest of the packet sent by Alice

    A report from a node on the reverse path $\theta_{ri} = [ri, \text{ack-digest}, \theta_{fi+1}]_i$ is ABOVE if
    - θ_{ri+1} is present, AND
    - The MAC on θ_{ri} is valid AND
    - the ack-digest field does *not* match $h([B, h(d)]_B)$, *i.e.* the hash of a valid ack for packet d.
  - Otherwise, a report is BELOW.

4.5 Statistical Fault Localization via a Composition Technique

Naive sampling is not useful: In contrast to the sampling technique we used to build secure statistical FD protocols, simply having Alice sample some packets as probes and running FL on only those probe packets does not make FL much more efficient. To see why, observe that Alice cannot inform intermediate nodes which packets she designates as probes (if she did, a node occupied by Eve could then violate probe indistinguishability, see Section 3.5). It follows that intermediate nodes must behave as through every packet they see is a probe, and the storage overhead at the intermediate nodes remains as in a per-packet FL protocol!

An insecure composition: Perhaps the most natural way to compose statistical FD protocols to obtain a statistical FL protocol (similar to the approach of [24, 4], see Figure 23-(a)) is to have Alice run N simultaneous probing protocols with each of the intermediate nodes, and use the statistics from each to infer behaviour at each link. Unfortunately, this composition is vulnerable to the following *timing attack*: Suppose an isolated burst of packets travels through the network. The burst will trigger a separate burst of acks as it reaches each intermediate node. It follows that Eve (as in Figure 23-(b)) can determine the origin of each ack by observing the *time* at which the ack arrives. Then, Eve can drop all the acks originating from an honest node (Bob), causing Alice to implicate a link $(3, B)$ which is not adjacent to Eve. Notice that this attack results from the structure of the composition in Figure 23, and cannot be prevented even when acks are encrypted.

A secure composition: We can use secure statistical FD protocols to construct a secure statistical FL via the composition method in Figure 24. We assume that every node R_i has public key PK_i that is known to everyone, and keeps secret the corresponding secret key SK_i. Each intermediate node runs a probing session with Bob, with one modification: whenever Bob decides to send an ack, Bob will address that ack to Alice *no matter whom the ack is destined for!* Each node forwards all acks

[22] For example, if after classifying the fault ACKs, Alice obtains $\theta_{r1} = BELOW, \theta_{r2} = BELOW, ..., \theta_{ri} = BELOW, \theta_{ri+1} = ABOVE, ..., \theta_{rM} = ABOVE, \theta_{fN} = ABOVE, ..., \theta_{f1} = BELOW$, localize the fault to link $(ri, ri + 1)$.

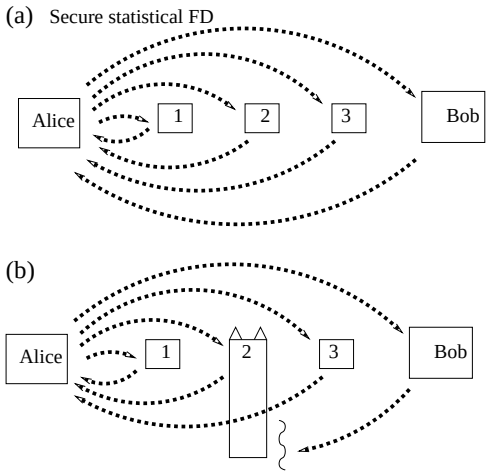

Figure 23: An insecure composition.

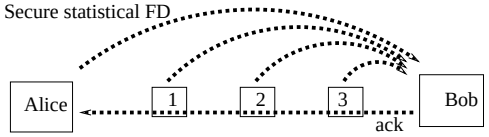

Figure 24: A secure composition.

upstream, and processes only the ack he expects. At the end of the FL session Alice will send a request $[A]_{SK_A}$, similar to the onion request of Section 4.4, to all the intermediate nodes. Then, upon node i's receipt of the onion report θ_{i+1} from downstream node $i+1$, node i responds with a digitally-signed onion report $\theta_i = [i, V_i, \theta_{i+1}]_{SK_i}$ where V_i is his *count* of the number of faults that occurred between himself and Bob.To prevent replay attacks, each node should also timestamp his onion report. To make the protocol monotone, each intermediate node R_i should only include the onion report from his downstream neighbour θ_{i+1} in his own report θ_i if the digital signature on θ_{i+1} is valid. When Alice receives the onion θ_1, she computes $F_\ell = \frac{V_i - V_{i+1}}{pT}$ for each link $\ell = (i, i+1)$, and adds ℓ to her list of faulty links if $F_\ell > \frac{\alpha_\ell + \beta_\ell}{2}$.

Security: Notice that now Eve cannot implicate an innocent link via (i) targeted dropping of acks, as in Figure 23-(b), because acks destined for different nodes are indistinguishable, or (ii) targeted dropping of reports, because the reports are onionized as in Section 4.4. In Section 4.5.1 and Section 4.5.2 we formally prove the security of composition with Pepper Probing and Salt Probing.

Choice of probing protocol: The composition works with either Pepper or Salt Probing. With Salt Probing, the communication overhead and the computations at Bob required for the composition are *identical* to those required for a single Salt Probing session between Alice and Bob, because the same set of packets are designated as probes (by Bob) for each intermediate node. Meanwhile, Pepper Probing increases communication overhead and processing at Bob (because each intermediate node needs its own set of acks) but less storage is required at Alice and the intermediate nodes (since Salt Probing always requires more storage than Pepper Probing). Composition via Pepper Probing also requires pairwise keys between Bob and each intermediate node, while composition via Salt Probing scales better, since it requires only that each node have a single public key.

Implications for routing architectures: We observe that architectures in which source networks make routing decisions by probing to intermediate nodes are also vulnerable to the timing attack we showed for the insecure composition. Our results imply that architectures in which intermediate nodes probe to destination networks are preferable from the perspective of security.

4.5.1 Statistical Fault Localization: Composition with Pepper Probing

As we described above, in Each party runs an independent copy of the Pepper protocol with Bob. Because of the two-message nature of Pepper (Alice sends the data packet and Bob replies with an ack), each intermediate node R_i does not inject additional messages but simply runs probing on the data it receives from R_{i-1}.

- Init: the Init function for Pepper composition generates public-private key pairs PK_i, SK_i for each $i \in \{1, \ldots, N, A, B\}$. It also generates PRF keys k_i^{mac}, which node R_i uses to perform coordinated secure sampling with Bob.
- R_i: each node R_i runs an independent Pepper protocol instance with R_B. There are two technical modifications to the scheme: first, Bob *addresses all acks to Alice* and not to the intermediate node that is expecting them. Second, at the end of the session, for each i, Alice sends an authenticated request $[A, T, \mathsf{nonce}]_{k_{A,i}}$ to each node R_i asking for its report, where nonce is chosen at random by Alice before she sends her authenticated request, and nonce is used to prevent replay attacks. Each node replies with the quantity $F_i = \frac{V_i}{pT}$, where V_i is the number of faulty exchanges that R_i witnessed. They send this data to Alice in an onion fashion: each node R_i generates a report $\theta_i = [i, \mathsf{nonce}, F_i, \theta_{i+1}]_{SK_i}$, where nonce is the same as the one Alice sent, and R_i replaces $\theta_{i+1} = \perp$ if it does not receive a report from R_{i+1} in a timely fashion, or if R_{i+1}'s report is invalidly signed (observe that R_i can verify R_{i+1}'s report because he knows R_i's public key).
- Decode: the identity.
- FaultFunction : Alice receives the onion report θ_1 and checks to make sure that all the layers of the onion are correctly MAC'd. If any layer of the onion is incorrectly MAC'd then Alice outputs the first link ℓ_0, between a node with an authentic report and a node with a forged report, and sets $F_{\ell_0} = 1$. Otherwise, for every link $\ell = (i, i+1)$, Alice computes $F_\ell = F_i - F_{i+1}$. Alice outputs link ℓ for each ℓ such that $F_\ell > \frac{\alpha_\ell + \beta_\ell}{2}$.

Theorem 4.30. *The the composition with Pepper Probing using a probe frequency of p, satisfies $(\vec{\alpha}, \beta, \delta, T_0)$-strong statistical security for the following T_0. Let $\alpha = \sum_\ell \alpha_\ell$, let $\beta_\ell = \alpha_\ell \frac{\beta}{\alpha}$ and let $\alpha_{\min} = \min_\ell \alpha_\ell$ and $\beta_{\min} = \min_\ell \beta_\ell$. Then we may set $T_0 = O(\frac{1}{p(\beta_{\min} - \alpha_{\min})^2} \ln \frac{N}{\delta})$.*

Proof. The probability that any efficient adversary Eve successfully forges an ack for a dropped packet by either finding a collision in the hash function or forging a MAC used in Pepper Probing is bounded by $\epsilon_{\mathsf{crh}} + \epsilon_{\mathsf{mac}}$. (See Section 3.4 the security proof for Per-Packet Ack for a more detailed argument of this.) Furthermore, the probability that any efficient adversary Eve successfully forges the onion report of an honest node (by forging the digital signature on the onion report) is bounded by $N\epsilon_{\mathsf{sig}}$. (The formal argument for this is identical to that given in Section 4.23 in the proof of the Optimistic per-packet FL Protocol.) As in the Optimistic per-packet FL Protocol of Section 4.4, if Eve tampers with the onion report, she immediately implicates a link adjacent to one of the nodes she controls. For the rest of the proof we condition on the event that Eve does not forge an ack to a dropped packet or validly forge the onion report of an honest node.

In the following we work within a single interval u conditioned on any previous history. Thus we omit u from the notation, *e.g.* we write κ_ℓ instead of $\kappa_\ell(u)$. This will not cause any problems in the analysis because we will work with a truly random function (and hence there is complete independence with previous intervals), and then use the PRF property to show that the same results hold when we replace the truly random function by a PRF.

We say that an exchange is *data-faulty* for R_i if the packet d that R_i sent was not received at Bob (it was either dropped or modified along the way). If we let $\mathcal{L}_{\mathsf{Data}}^{\mathsf{Bob}}$ denote the list of data packets received by Bob and $\mathcal{L}_{\mathsf{Data}}^{i}$ denote the list of data packets sent by R_i, then $D_i = |\mathcal{L}_{\mathsf{Data}}^{i} \setminus \mathcal{L}_{\mathsf{Data}}^{\mathsf{Bob}}|$. We say an exchange is *ack-faulty* if the data arrived at Bob correctly, but the exchange was a probe exchange and the corresponding ack was not received by Alice (it was dropped or modified between Bob and R_i for some i).

We let D_i denote the actual number of data-faulty exchanges between node R_i and Bob. (*i.e.* a data packet was dropped somewhere on the path between R_i and Bob). We let C_i denote the actual number of ack-faulty exchanges between node R_i and Bob (*i.e.* an ack packet was dropped somewhere on the path between R_i and Bob). Note that when R_i computes the number of faulty exchanges she sees, she is unable to distinguish between data-faulty exchanges and ack-faulty exchanges. Notice that $C_i = C_{i+1}$ if both $i, i+1$ are uncorrupted. However, it is possible that $D_i > D_{i+1}$ for two adjacent uncorrupted nodes because of congestion. We let $p' = \frac{p}{1-(1-p)^N}$, which is the probability that a particular node R_i sampled an exchange given that at least one node sampled that exchange.

The proof of the theorem hinges on the following lemma (which we prove later), that says that Alice's estimate is close to a weighted sum of the fraction of data-faulty and ack-faulty exchanges. Let us set $\gamma = \frac{\beta_{\min} - \alpha_{\min}}{2}$:

Lemma 4.31. *For each $i \notin S$ where S is the set of nodes corrupted by Eve, with probability $> 1 - \frac{\delta}{2(N+1)} - N\epsilon_{\mathsf{prf}}$, it holds that*

$$|F_i - \tfrac{1}{T}(D_i + \tfrac{p'}{p}C_i)| \leq \tfrac{1}{2}\gamma$$

Proving the correctness condition: We now show that Eve cannot bias Alice's estimate of the fault rate for a link between two honest nodes. To do this, we show that Lemma 4.31 implies that the probability that there exists a link $\ell = (i, i+1)$ that is not adjacent to Eve such that $|\kappa_\ell - F_\ell| > \gamma$ is bounded by at most $\delta + 2N(N+1)\epsilon_{\mathsf{prf}}$. To see why, observe that for each link the probability that either honest node R_i's local fault estimate F_i is skewed, *i.e.*

$$|F_i - \tfrac{1}{T}(D_i + \tfrac{p'}{p}C_i)| > \tfrac{1}{2}\gamma$$

or honest node R_{i+1}'s local fault estimate F_{i+1} is skewed, *i.e.*

$$|F_{i+1} - \tfrac{1}{T}(D_{i+1} + \tfrac{p'}{p}C_{i+1})| > \tfrac{1}{2}\gamma$$

occurs with probability bounded by at most $\frac{\delta}{(N+1)} + 2N\epsilon_{\mathsf{prf}}$. (This holds because of a union bound over all uncorrupted links, where there can be at most $N+1$ uncorrupted links).

Conditioning on the event that both R_i's or R_{i+1}'s local fault estimate is *not* skewed, we can then derive the following. For an uncorrupted link $\ell = (i, i+1)$, from the definition of κ_ℓ, it follows that $\kappa_\ell = \frac{1}{T}(D_i - D_{i+1})$. Next we can write:

$$|F_\ell - \kappa_\ell| = |F_i - F_{i+1} - \tfrac{1}{T}(D_i - D_{i+1})|$$

Using the fact that $C_i = C_{i+1}$, we get:

$$\begin{aligned}
|F_\ell - \kappa_\ell| &= |F_i - F_{i+1} - \tfrac{1}{T}(D_i + \tfrac{p'}{p}C_i - (D_{i+1} + \tfrac{p'}{p}C_{i+1}))| \\
&\leq |F_i - \tfrac{1}{T}(D_i + \tfrac{p'}{p}C_i)| + |F_{i+1} - \tfrac{1}{T}(D_{i+1} + \tfrac{p'}{p}C_{i+1})| \\
&\leq \gamma
\end{aligned}$$

with probability at least $1 - \delta - 2N(N+1)\epsilon_{\mathsf{prf}}$. Since $\gamma = \frac{\beta_{\min} - \alpha_{\min}}{2}$, in particular this means that if $\kappa_\ell < \alpha_\ell$ then it holds that $F_\ell < \frac{\alpha_\ell + \beta_\ell}{2}$ and so Alice outputs $\mathsf{NoFault}$.

Proving the security condition: We now show that, if Eve drops more than a β fraction of packets in any interval, then Alice will catch her with probability at least $1 - \delta$.

Again suppose that the good event of Lemma 4.31, namely $|F_i - \frac{1}{T}(D_i + \frac{p'}{p}C_i)| \leq \frac{\gamma}{2}$ holds for all i, which happens with probability $> 1 - \delta - 2N(N+1)\epsilon_{\mathsf{prf}}$.

Since $\kappa_A = \frac{1}{T}D_A > \beta$, in particular this means that $F_A > \beta - \frac{\gamma}{2}$. But now by the telescoping property of the fault estimator we have that

$$F_A = \sum_{i=0}^{N}(F_{i+1} - F_i)$$

where we have identified $A = 0, B = N + 1$. We now claim, that by an averaging argument, that there exists some i such that $F_{i+1} - F_i > \frac{\alpha_\ell + \beta_\ell}{2}$ where link $\ell = (i, i+1)$. To see why, suppose the contrary, that for all i we had $F_{i+1} - F_i \leq \frac{\alpha_\ell + \beta_\ell}{2}$. Then, it follows that

$$F_A = \sum_{i=0}^{N} (F_{i+1} - F_i) \leq \sum_{\ell} \frac{\alpha_\ell + \beta_\ell}{2} = \frac{\alpha + \beta}{2} < \beta - \frac{\gamma}{2}$$

which is a contradiction of our condition that $F_A > \beta - \frac{\gamma}{2}$. As such, there is at least one link $\ell = (i, i+1)$ such that $F_\ell > \frac{\alpha_\ell + \beta_\ell}{2}$. Now from the proof of the correctness condition we know that this can't be any of the uncorrupted links, because the estimates F_ℓ for ℓ a good link is going to satisfy $F_\ell - \kappa_\ell < \frac{\gamma}{2}$ and in particular $F_\ell \leq \frac{\alpha_\ell + \beta_\ell}{2}$ for good ℓ. Therefore, it must be one of the bad links that was implicated, and so overall, with probability at least $1 - \delta + 2N(N+1)\epsilon_{\mathsf{prf}}$, when Eve cheats by more than a $\kappa > \beta$ fraction, a corrupted link is localized, and the protocol is secure. ■

It only remains to prove Lemma 4.31.

Proof of Lemma 4.31. Consider the random variable F_i' which is generated as F_i is generated by the Pepper scheme, except that we assume that instead of sampling according to a pseudo-random function $f_{k_i^{\mathsf{mac}}}$ Alice and Bob sample according to a shared truly random function ϕ_i. This means that, regardless of how Eve behaves, every packet that she drops was a probe for each R_i with probability p independent of her actions and each other, and every ack that she drops was an ack for each R_i with probability p' independent of her actions and each other. This follows from the fact that in this composition of Figure 24-(a) acks intended for different nodes are indistinguishable (unlike in the insecure composition of Figure 23)!

We will show now that this means F_i' is the average of many independent random variables. Let $S_{D_i}, S_{C_i} \subseteq [T]$ denote the exchanges that are data-faulty and ack-faulty, respectively (we can order the exchanges in an arbitrary way, say in the order in which the first message of the exchange was sent), and notice hat $S_{D_i} \cap S_{C_i} = \varnothing$ since an exchange cannot be both data- and ack-faulty. We now define the random variables X_t for $t \in [T]$. For $t \notin S_{D_i} \cup S_{C_i}$, the variable X_t is identically 0. Otherwise, X_t is defined as follows:

$$\text{For } t \in S_{D_i},\ X_t = \begin{cases} 1 & \text{w.p. } p \\ 0 & \text{w.p. } 1 - p \end{cases}$$

$$\text{For } t \in S_{C_i},\ X_t = \begin{cases} 1 & \text{w.p. } p' \\ 0 & \text{w.p. } 1 - p' \end{cases}$$

We claim that $F_i' = \frac{1}{pT} \sum_{t=1}^{T} X_t$ because sampling is done using a truly random function and all the sample points (*i.e.* the data packets) are unique so each packet is sampled independently with probability p. Thus each data-faulty exchange in S_{D_i} was sampled by R_i with probability p, while each ack-faulty exchange in S_{C_i} was sampled by R_i with probability p' (because here we need to condition on the fact that at least one node sampled the ack-faulty exchange (so that Bob generates an ack for that exchange!)).

But it is clear then that $\mathbb{E}[F_i'] = \frac{1}{T}(D_i + \frac{p'}{p} C_i)$:

$$\Pr[|F_i' - \tfrac{1}{T}(D_i + \tfrac{p'}{p} C_i)| > \gamma] \leq \Pr[|pF_i' - \tfrac{1}{T}(pD_i + p'C_i)| > p\gamma] \tag{4.6}$$

$$= \Pr[\tfrac{1}{T} |\sum_t X_t - (pD_i + p'C_i)| > p\gamma] \tag{4.7}$$

Letting $\mu = \frac{1}{T}(pD_i + p'C_i)$:

$$= \Pr[|\tfrac{1}{T} \sum_t X_t - \mu| > \tfrac{p\gamma}{\mu} \mu] \tag{4.8}$$

At this point we would like to say that $\mu = O(p)$ and use p in a bound on Equation 4.8. However, using a crude bound only gives us $\mu = O(p')$. Instead, in the rest of this proof we shall carefully show that that the probability that $C_i > 2\frac{p}{p'}T$ is at most $\frac{\delta}{8(N+1)}$, and conditioned on $C_i \leq 2\frac{p}{p'}T$ then we also have that the probability that $|\frac{1}{T}\sum_t X_t - \mu| > p\gamma$ is bounded by $\frac{\delta}{8(N+1)}$, which gives us an overall bound of $\frac{\delta}{4(N+1)}$.

We start by letting Y denote the number of exchanges in the game that require acknowledgements for any R_i. Notice that $\mathbb{E}[Y] = \frac{p}{p'}T$ and that $C_i \leq Y$ unconditionally, simply because one can't tamper with an ack if it was never sent. Now we split the probability in Inequality 4.8 as follows:

$$\begin{aligned}
\Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu] &\leq \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \text{ and } C_i > 2\tfrac{p}{p'}T] + \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \text{ and } C_i \leq 2\tfrac{p}{p'}T] \\
&\leq \Pr[C_i > 2\tfrac{p}{p'}T] + \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \;\mid\; C_i \leq 2\tfrac{p}{p'}T] \\
&\leq \Pr[Y > 2\tfrac{p}{p'}T] + \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \;\mid\; C_i \leq 2\tfrac{p}{p'}T]
\end{aligned}$$

Now by a Chernoff bound the first probability is at most $\frac{\delta}{8(N+1)}$ for $T = O(\frac{p'}{p}\ln(\frac{N}{\delta})) \leq O(\frac{1}{p}\ln(\frac{N}{\delta}))$, since Y is the sum of independent $\frac{p}{p'}$-biased random variables.

The second probability can now be bounded by a Chernoff bound because notice that, although the *definition* of the distributions of the X_t depends on C_i, the actual randomness of the X_t is independent of C_i. This gives us that the second probability is bounded by

$$\Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \;\mid\; C_i \leq 2\tfrac{p}{p'}T] \leq 2^{-\Omega(\frac{p^2\gamma^2}{\mu}T)} \tag{4.9}$$

and now because of the conditioning we can bound

$$\mu = \tfrac{1}{T}(pD_i + p'C_i) \leq \tfrac{1}{T}(pD_i + 2pT) \leq 3p$$

and so we can plug into Inequality 4.9 and set $T = O(\frac{1}{p\gamma^2}\ln(N/\delta))$ appropriately to get

$$\Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{p\gamma}{\mu}\mu \;\mid\; C_i \leq 2pT] \leq 2^{-\Omega(p\gamma^2 T)} \leq \tfrac{\delta}{8(N+1)}$$

Combining this with the previous bound on $\Pr[Y > 2\frac{p}{p'}T]$ gives us that

$$\Pr[|F_i' - \tfrac{1}{T}(D_i + \tfrac{1}{p}C_i)| > \gamma] \leq \tfrac{\delta}{4(N+1)}$$

To complete the proof of the lemma, it suffices to observe that if replacing a truly random function ϕ_i with a PRF $f_{k_i^{\mathsf{mac}}}$ alters the probability by more than $N\epsilon_{\mathsf{prf}}$, then we can efficiently distinguish between ϕ_i and $f_{k_i^{\mathsf{mac}}}$ with advantage ϵ_{prf} by using the distinguisher that simply simulates this entire game, using access to an oracle containing either ϕ_i or $f_{k_i^{\mathsf{mac}}}$ to answer calls to the PRF from the scheme, and then outputting 1 iff the condition $|F_i - \frac{1}{T}(D_i + \frac{p'}{p}C_i)| \leq \frac{1}{2}\gamma$ is violated.[23] ■

4.5.2 Statistical Fault Localization: Composition with Salt Probing

The composition of Salt Probing is similar to that with Pepper Probing. However, instead of each party running an independent copy of the Salt Probing with Bob, they all "share the same instance of the protocol", since each ack that Bob sends is relevant to every intermediate node (up to to subsampling by

[23] This distinguisher in fact requires access to N oracles, either all computing either a truly random function or all computing a PRF. Then we can turn this into a distinguisher for a single oracle using the hybrid argument, which is why we lose a factor of N in the distinguishing advantage. See *e.g.* [12] for details about this kind of argument.

individual intermediate nodes). Because of the two-message nature of Salt (Alice sends the data packet and Bob replies with an ack), each intermediate node R_i does not inject additional messages but simply runs probing on the data it receives from R_{i-1}.

Furthermore, the composition with Salt Probing runs completely in the public-key setting. More formally,

- Init: the Init function for Salt composition generates a public/private signing key pair PK_i, SK_i for each $i \in [N] \cup \{A, B\}$. The public keys are distributed to each R_i while the secret keys are kept secret.
- R_i: each node R_i runs the same Salt protocol instance with R_B. In each salt interval, Bob picks a random salt pair (s_1, s_2) and uses s_1 to key a PRF and s_2 to key a secure MAC. Bob samples and acks probes in the same way as in usual Salt Probing, but when he sends acks he *addresses them all to Alice* and not to any intermediate node. Second, at the end of each salt interval, Bob sends a salt release message containing $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$ so that each intermediate node can compute its estimates V_i of the number of faulty exchanges. Finally, at the end of the entire session, Alice sends an authenticated request $[A, T, \mathsf{nonce}]_{SK_A}$ to all nodes R_i asking for their reports, where nonce is again used to prevent replay attacks. The nodes respond with an onion report exactly as in the composition of Pepper Probing, including their observed fault information $F_i = \frac{V_i}{pqT}$.
- Decode: The identity.
- FaultFunction : This is identical to the FaultFunction function of the composed Pepper protocol.

To prove the security of the composition with Salt Probing, we first argue that by security of Salt Probing (see Section 3.7) Eve cannot (with better than negligible probability) break security by manipulating multiple salt intervals. In particular, in the argument in Theorem 3.22, the fact that (i) Eve cannot skew Alice's (or any intermediate node's) synchronization to Bob's clock, and (ii) Eve cannot forge a Salt release message, apply in the composition setting as well. It follows that if Eve wants to break security, she must bias Alice's measurements during a single salt interval. We now give a rigorous proof that within a single salt interval, Eve is unable to bias Alice's measurements.

Theorem 4.32. *Composed salt probing with frequency p and subsampling rate q satisfies $(\vec{\alpha}, \beta, \delta, T_0)$-strong statistical security for FL for the setting of*

$$T_0 = O(\frac{1}{qp(\beta_{\min} - \alpha_{\min})^2} \ln \frac{N}{\delta})$$

where $\alpha_{\min}, \beta_{\min}$ are defined as in Theorem 4.30.

Proof. The proof follows essentially the same pattern as the proof of composition with Pepper Probing. Again, let D_i, C_i denote the number of data-faulty and ack-faulty exchanges occurring between R_i and Bob. Let $\gamma = \frac{\beta_{\min} - \alpha_{\min}}{2}$.

We use a similar lemma (which we prove later):

Lemma 4.33. *For each $i \notin S$ where S is the set of nodes corrupted by Eve, with probability $> 1 - \frac{\delta}{4(N+1)} - N\epsilon_{\mathsf{prf}}$, it holds that*

$$|F_i - \tfrac{1}{T}(D_i + \tfrac{1}{p}C_i)| \leq \tfrac{1}{2}\gamma$$

Using Lemma 4.33, the proof that the statistics for the *good links* and for the *corrupted links* cannot be biased by Eve proceeds in exactly the same way as in the composition with Pepper Probing.

In the proof there is the question of independence between the Salt Probing sessions between each R_i and Bob. Whereas in the composition with Pepper Probing where the individual sessions are independent of each other, in the composition with Salt Probing the individual sessions are dependent because they use the same salt value. However, this does not affect our analysis because we bound the probability of Eve succeeding with union bound, so that the dependence is immaterial. ■

Proof of Lemma 4.33. The proof is nearly identical as that of Lemma 4.31, except that in this case we need to take into account subsampling. Again letting F_i' denote the observed fault rates at each node R_i, and again using the same sets $S_{D_i}, S_{C_i} \subseteq [T]$ denoting the data-faulty and ack-faulty exchanges, and defining the random variables

$$\text{For } t \in S_{D_i},\ X_t = \begin{cases} 1 & \text{w.p. } pq \\ 0 & \text{w.p. } 1-pq \end{cases}$$
$$\text{For } t \in S_{C_i},\ X_t = \begin{cases} 1 & \text{w.p. } q \\ 0 & \text{w.p. } 1-q \end{cases}$$

we see similar to before that $F_i' = \frac{1}{pqT}\sum_{t=1}^{T} X_t$. Applying the fact that $\mathbb{E}[F_i'] = \frac{1}{T}(D_i + \frac{1}{p}C_i)$, we can again compute:

$$\begin{aligned}\Pr[|F_i' - \tfrac{1}{T}(D_i + \tfrac{1}{p}C_i)| > \gamma] &\leq \Pr[|pqF_i' - \tfrac{1}{T}(pqD_i + qC_i)| > pq\gamma] \\ &= \Pr[\tfrac{1}{T}|\sum_t X_t - (pqD_i + qC_i)| > pq\gamma]\end{aligned}$$

Letting $\mu = \frac{1}{T}(pqD_i + qC_i)$:

$$= \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{pq\gamma}{\mu}\mu]$$

Doing a similar conditioning as in the proof of Pepper, except this time conditioning on $C_i \leq 2pT$ since all the nodes share the same acks, gives us that this last probability is bounded by

$$\begin{aligned}&\leq \Pr[C_i > 2pT] + \Pr[|\tfrac{1}{T}\sum_t X_t - \mu| > \tfrac{pq\gamma}{\mu} \ \mid\ C_i \leq 2pT] \\ &\leq 2^{-\Omega(pT)} + 2^{-\Omega(pq\gamma^2 T)} \\ &\leq \tfrac{\delta}{4(N+1)}\end{aligned}$$

for an appropriate choice of $T = O(\frac{1}{pq\gamma^2}\ln(\frac{N}{\delta}))$.

Now the lemma can be completed by the observation that for any PRF family, the probability for the event $|F_i - \frac{1}{T}(D_i + \frac{1}{p}C_i)| > \gamma$ is $N\epsilon_{\mathsf{prf}}$ close to the same event for F_i'. ■

5 Conclusions and Implications

We conclude with some thoughts about the implications of our results on network architecture.

Fault Detection? Because FD protocols require only pairwise participation from nodes, deployment of FD can proceed in an incremental fashion that is compatible with incentives for informing routing decisions at the network edge. However, when we consider the placement and selection of FD protocols, natural questions arise about the division of labour between the end host and the edge router. We argue that the placement of FD protocols depends on the parties responsible for providing confidentiality and driving routing decisions. Consider, for example, the following three scenarios:

- Firstly, if the end host both provides confidentiality and makes routing decisions, then host-based cryptography, *e.g.* SSL, is appropriate for detecting faults on a per-packet basis. This may be the case in a new routing architecture, or a special-purpose secure network.
- On the other hand, if the edge router both provides confidentiality and makes routing decisions, then Stealth Probing [3]'s secure statistical FD protocol, that also performs edge-to-edge encryption of traffic, is most appropriate. This particularly appropriate when two stub ASes belong to the same institution.
- Finally, when routers make routing decisions but confidentiality is optionally provided at the end hosts, our Pepper and Salt Probing protocols (designed to avoid modifying and re-encrypting traffic at the edge router) are most appropriate. We believe that this is the case for the majority of scenarios at the edge of the Internet (*e.g.* a residential broadband provider who runs FD on traffic that users optionally protect with SSL), as well as in the *core* of Internet.

In fact, our Pepper and Salt Probing may even be efficient enough to be deployed in the core of Internet, as part of an architecture where core routers inform their routing decisions by running FD to destination networks.

Fault localization? FL protocols are likely to be practical for special settings, *e.g.* highly-secure networks owned by a single party, or for a subset of packets within an AS, *e.g.* network-management traffic. However, because FL requires active participation from the intermediate nodes, basing a network accountability framework on FL is not obviously compatible with the deployment incentives of the intermediate nodes on the Internet. (Recall that FL requires participation, *i.e.* dedicated storage, at each intermediate node even in a weaker security model where Eve can only selectively drop packets.) Our negative results imply that reducing the complexity of FL is only possible under a weaker security model: for example, by assuming some trust between ASes, or by detecting bad behaviour by using additional out-of-band information about the content of the packets, or by considering an adversary with strictly weaker abilities. We leave a rigorous treatment of these alternatives for future exploration.

Accountability? Our FD protocols may be a more appropriate building block for constructing an accountability framework based on bilateral business relationships between ASes. For instance, a stub AS may blame poor edge-to-edge performance on its immediate provider who could, in turn, ascribe blame to the next AS in the path, as suggested in [19]. (Interestingly, the structure of our statistical FL protocol echos that of [19]'s accountability framework). Another possible accountability framework could combine multiple FD measurements from different vantage points to localize faulty links and nodes in the network, in the style of network tomography. However, traditional approaches to network tomography based on maximum likelihood estimation [6, 7], do not apply when an adversary is actively trying to evade detection. We believe that this open problem would benefit from a rigorous approach, similar to the one we took in this paper, that combines theoretical cryptography with statistics to construct secure protocols and establish negative results for the adversarial setting.

A Learning transcripts

Naor and Rothblum showed that, if (infinitely often) one-way functions do not exist, then every adaptively changing distribution (ACD) is learnable in polynomial time. We use their result to prove that each node must perform some cryptographic operations in a secure FL protocol. See Section 4.3.

[22] define an ACD as a pair (G, D) of random processes. G is the generation process that takes a uniform string x and generates an initial state $s = G(x)$. It is useful to think of this as a key generation algorithm. D is a process that takes s and a history and uses them to generate a string. It is useful to think of D as generating a transcript for a protocol between two parties. The history consists of tuples (r_i, τ_i) for each of its past transcripts where r_i was the random string used to generate the transcript τ_i. The τ_i are the outputs of the ACD, while s and r_i remain secret. We will consider only ACD's that are efficiently constructible, *i.e.* the processes G, D are computable in polynomial time.

Definition A.1. We say that an ACD (G, D) is (δ, ε)-learnable in polynomial time if there exists an algorithm L satisfying the following:

1. L sees at most $m = \text{poly}(n)$ samples from (G, D). Let $\tau_1, \ldots, \tau_m$ denote these samples.
2. L generates a sample that, with probability $\geq 1 - \delta$ is ε-close to the distribution of what D would generate conditioned on the past transcripts $\tau_1, \ldots, \tau_m$.

Note that the conditioning in the second item above is only over τ_i and not over (r_i, τ_i). For details about this definition, please see [22]. Naor and Rothblum [22] proved the following theorem.

Theorem A.2. *If (infinitely often)-OWF do not exist, then every ACD is (δ, ε)-learnable in time $O(\frac{n}{\delta^2 \varepsilon^4})$.*

The result of [22] allows an adversary (that can break the security of a an (infinitely often)-OWF) to impersonate an transcript produced by a single entity D. Notice that this lemma does not consider interaction between parties that generated the transcript; instead [22] assume that the transcript produced by a single entity D. We extend their result to the interactive protocols. Consider two interactive algorithms A and B. In the following lemma, we show how an adversary B' (that can break the security of a an (infinitely often)-OWF) can impersonate honest algorithm B while interacting with honest algorithm A.

Lemma A.3. *Let $\langle A, B \rangle$ denote the transcript of a protocol between two parties algorithms A, B, which are independent randomized algorithms. Let $\langle A', B' \rangle$ be the transcript using (different) independent randomized algorithms A', B'.*[24] *If the protocol lasts ℓ rounds and the statistical difference between the two is bounded by ε,* i.e. *$\Delta(\langle A, B \rangle, \langle A', B' \rangle) \leq \varepsilon$, then we have $\Delta(W \circ \langle A, B' \rangle, W \circ \langle A, B \rangle) \leq \ell\varepsilon$ where W denotes the internal random coins of A used to create the respective transcripts.*

Proof of Lemma A.3. Let A_i be the i'th next-message function of A, which we assume to be deterministic; we will make explicit any randomness used by A later in the proof. Let B_i be the i'th next-message function of B, which may be randomized. We assume *wlog* that the first message in the protocol is sent by A and the last message is sent by B (if not, just send empty strings).

We recursively define σ_i, which will represent the partial transcripts of the protocol, by the following:

$$\begin{aligned} \sigma_0 &= \perp \\ \sigma_{2i} &= \sigma_{2i-1} \circ B_i(\sigma_{2i-1}) \\ \sigma_{2i+1} &= \sigma_{2i} \circ A_i(w, \sigma_{2i}) \end{aligned}$$

[24] Our assumption that A, B are independent and A', B' are independent is true of any protocol realizable in the real world. This is in contrast to protocols that assume a random oracle or an ideal cipher, where A, B and A', B' have access to shared (and hence dependent) randomness.

where $\perp$ denotes the empty string. Note that the σ_i are random variables (because the B_i may be randomized). Similarly define σ'_i except with A, B replaced by A', B'. Lastly, define σ_i^{alt} by

$$\begin{aligned}\sigma_0^{\mathsf{alt}} &= \perp \\ \sigma_{2i}^{\mathsf{alt}} &= \sigma_{2i-1}^{\mathsf{alt}} \circ B'_i(\sigma_{2i-1}) \\ \sigma_{2i+1}^{\mathsf{alt}} &= \sigma_{2i}^{\mathsf{alt}} \circ A_i(w, \sigma_{2i})\end{aligned}$$

i.e. the partial transcripts of B' interacting with A. Note that the hypothesis of our lemma states that $\Delta(\sigma_\ell, \sigma'_\ell) \leq \varepsilon$, and since the σ_i are substrings of σ_ℓ it follows that for all i we have $\Delta(\sigma_i, \sigma'_i) \leq \varepsilon$.

We want to show that $\Delta(W \circ \sigma_\ell, W \circ \sigma_\ell^{\mathsf{alt}}) \leq \ell\varepsilon$ where the randomness is over W, and the randomness of the B_i. This is shown by induction. We shall show that each message in the interaction between A and B' adds ε statistical distance from the honest transcript. (Thus, since the interaction has ℓ messages, the total statistical distance between the honest transcript and the impersonated transcript is $\ell\varepsilon$).

To start the induction, we have that $\Delta(W \circ \sigma_0, W \circ \sigma_0^{\mathsf{alt}}) = 0$. To show the inductive step, we show that $\Delta(W \circ \sigma_{2i}, W \circ \sigma_{2i}^{\mathsf{alt}}) \leq 2i\varepsilon$ implies that $\Delta(W \circ \sigma_{2i+2}, W \circ \sigma_{2i+2}^{\mathsf{alt}}) \leq (2i+2)\varepsilon$.

By the inductive hypothesis and the fact that applying the same random process (or deterministic process) to a random variable never increases statistical distance, we have that

$$\Delta(W \circ \sigma_{2i+1}^{\mathsf{alt}}, W \circ \sigma_{2i+1}) = \Delta(W \circ \sigma_{2i}^{\mathsf{alt}} \circ A_i(W, \sigma_{2i}^{\mathsf{alt}}), W \circ \sigma_{2i} \circ A_i(W, \sigma_{2i})) \leq 2i\varepsilon \tag{1.1}$$

We consider $\Delta(\sigma_{2i+1}, \sigma'_{2i+1}) \leq \varepsilon$. Although we usually think of $W \circ \sigma_{2i+1}$ as first sampling W and then computing σ_{2i+1} (with the aid of interacting with B), one can also view it as taking a sample of σ_{2i+1} and then sampling a random W consistent with σ_{2i+1}. Let $W(\sigma)$ be a random string from the space of all strings that would produce a particular transcript σ using A, B (set $W(\sigma) = 0$ if σ could not possibly have been produced by A, B). Now, notice that $W(\sigma)$ is well-defined also when σ is drawn according to σ'_{2i+1}, and therefore that

$$\Delta(\sigma_{2i+1}, \sigma'_{2i+1}) \leq \varepsilon \quad \Rightarrow \quad \Delta(W(\sigma_{2i+1}) \circ \sigma_{2i+1}, W(\sigma'_{2i+1}) \circ \sigma'_{2i+1}) \leq \varepsilon$$

Applying this and the triangle inequality to Inequality 1.1 we get

$$\Delta(W \circ \sigma_{2i+1}^{\mathsf{alt}}, W(\sigma'_{2i+1}) \circ \sigma'_{2i+1}) \leq (2i+1)\varepsilon$$

Applying B'_i to both terms again does not increase statistical distance because B' is independent from A'. Furthermore $W(\sigma'_{2i+2}) = W(\sigma'_{2i+1})$ because B is independent of A, which gives us

$$\Delta(W \circ \sigma_{2i+2}^{\mathsf{alt}}, W(\sigma'_{2i+2}) \circ \sigma'_{2i+2}) \leq (2i+1)\varepsilon$$

This is our crucial application of the independence of A, B and A', B'. *The above step is false if independence does not hold.* For example, in the case where A, B and A', B' share an ideal cipher the above does not hold.

Finally applying $\Delta(\sigma_{2i+2}, \sigma'_{2i+2}) \leq \varepsilon$ and the same reasoning as above gives us

$$\Delta(W \circ \sigma_{2i+2}, W \circ \sigma_{2i+2}^{\mathsf{alt}}) \leq (2i+2)\varepsilon$$

as desired. ■

B Necessity of Cryptography for Fault Detection: Reduction to One-Way Functions

In Section 3.3.2 proved the necessity of cryptography for secure fault detection by using a reduction from secure FD to keyed identification schemes (KIS). We choose to use this proof because it is more intuitive, and also because it shows that 'secure identification' is an integral part of secure FD. In this section we prove the necessity of cryptography for secure FD by using a reduction from secure FD to one way functions. While this reduction gives less intuition about the nature of secure FD, it is in some sense more standard. One way functions are widely viewed as the fundamental primitive of cryptography. Since the work of Impagliazzo and Luby [16], reductions to one-way-functions have been used to prove the necessity of cryptography.

B.1 Per-packet Fault Detection

First we show that we can construct a one-way function from any weakly per-packet secure fault detection scheme, which implies that weakly per-packet secure fault detection is at least as strong as OWF's and most basic (symmetric) cryptographic primitives. Using Lemma A.3 and Theorem A.2 we prove the following.

Proposition B.1. *If there exists a per-packet secure FD scheme, then there exists i.o.-OWF.*

Proof. We prove this in the contrapositive. Assume that there exists an efficient adversary, Ivan (the inverter), that can invert an i.o. OWF. We shall show that the existence of Ivan implies that there exists an Eve that can break the security of a per-packet FD protocol.

From Theorem A.2, we know that the existence of Ivan, that can invert an i.o.-OWF, implies that every ACD is (δ, ε)-learnable in polynomial time. Consider the ACD given by the following: G is Init and D is the computation that generates the transcript $\langle R_A, R_B\rangle(d)$ for a uniformly random message d.

Since every ACD is learnable, in particular Eve can learn how to reproduce an exchange that is ε-close to a true exchange (by using the Ivan with the learning algorithm of [22]). That is, from Theorem A.2, the existence of Ivan implies that there exists an efficient algorithm L that can find, with probability $3/4$, an x' such that $\overrightarrow{\mathsf{aux}'} = \mathsf{Init}(x')$ and

$$\Delta(\langle R_A(\mathsf{aux}'_A), R_B(\mathsf{aux}'_1)\rangle(d), \langle R_A, R_B\rangle(d)) \leq \varepsilon$$

where we set $\varepsilon = \frac{1}{2\ell}$ where ℓ is an upper bound on the number of rounds in the fault detection protocol (*e.g.* Per-Packet Ack is a 2-round FD protocol).

Now applying Lemma A.3 setting A to be Alice and B to be the honest Bob, B' to be $R_B(\mathsf{aux}'_B)$, it follows that

$$\Delta(\langle R_A, R_B(\mathsf{aux}'_B)\rangle(d), \langle R_A, R_B\rangle(d)) \leq \ell\varepsilon \leq 1/2$$

So now an adversary Eve can simply sit on the path between Alice and Bob, and obtain enough transcripts so that she may use L to learn how to impersonate Bob, and then on the next exchange Eve can drop the packet and respond to Alice's messages using L. With probability $\geq 3/4$ she generates a x' that is ε-close to the true x, so with probability $3/4 - \ell\varepsilon \geq 1/4$ Eve convinces Alice that nothing was wrong in this last exchange, thus breaking weak per-packet security. Looking more closely at the result of [22], we see that the learning algorithm will run in time $O(n\ell^4)$. ■

B.2 Statistical Fault Detection

We now show that any scheme that statistically secure must also be as strong as OWF. This proof also uses the techniques of [22] discussed in Section A, but is more involved because the statistical guarantee we desire is much weaker than the per-packet guarantee.

The main difference is that Eve now needs to successfully cheat *many times*, whereas in the per-packet case it suffices for her to cheat just once. The effect of this on the proof is in the parameters: instead of learning a single exchange, we will now ask that Eve learn an entire interval of T exchanges. This will cause a significant deterioration in the parameters: instead of running for $O(n\ell^2)$ exchanges, Eve will have to run for $O(nT_0^2\ell^2)$ exchanges. (Recall that ℓ is the number of rounds in each exchange of the FD protocol, *e.g.* Per-Packet-ACK is a 2-round protocol, and T_0 is the number of exchanges in each probing interval.)

Theorem B.2. *If there exists a $(\alpha, \beta, \delta, T_0)$-statistically secure FD scheme (for any function T_0 of α, β, δ), then there exists i.o.-OWF.*

Proof. This proof is entirely identical to the proof of the per-packet case with the following change to the ACD; G is Init as before, but D is now the transcript of an entire *interval* of T_0 exchanges, rather than of a single exchange.

Whereas before we showed that we can generate the transcript of a new exchange using aux' that is ε-close to an exchange generated by aux, in this case we can generate the transcript of a new *interval* of T_0 exchanges that is ε-close to an interval generated by aux. Now, applying Lemma A.3 says that in the next interval, call it interval u, when Eve drops all the data in that interval (thus causing $\kappa(u) = 1$) and uses aux' to impersonate B to A, she generates a transcript that is $T_0\ell\varepsilon$ close to a true interaction, and so convinces Alice that no faults occurred with probability $1 - T_0\ell\varepsilon$. Therefore, by setting $\varepsilon = O(\frac{1}{T\ell})$, we get an algorithm running in time $O(n(T_0\ell)^4)$ that has probability $> 1/2$ of breaking security. ■

References

[1] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing packet obituaries. *ACM HotNets-III*, 2004.

[2] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *IEEE INFOCOM*, 2004.

[3] I. Avramopoulos and J. Rexford. Stealth probing: Efficient data-plane security for ip routing. *USENIX*, 2006.

[4] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *ACM WiSE*, 2002.

[5] F. Baccelliand, S. Machiraju, D. Veitch, and J. Bolot. The role of PASTA in network measurement. In *ACM SIGCOMM*, 2006.

[6] R. Caceres, N. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Trans. Info. Theory*, 45(7), 1999.

[7] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical Science*, 19(3):499–517, 2004.

[8] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. *IEEE/ACM Trans. Netw.*, 13(3), 2005.

[9] M. Crovella and B. Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. Wiley, 2006.

[10] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *ACM SIGCOMM*, 2000.

[11] S. Goldberg, D. Xiao, B. Barak, and J. Rexford. Measuring path quality in the presence of adversaries: The role of cryptography in network accountability. *Working paper*.

[12] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2007.

[13] D. Harkins and D. Carrel. Rfc2409: The internet key exchange (ike).

[14] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–96, 2001.

[15] IETF. Working Group on IP Performance Metrics (ippm). *http://www.ietf.org/html.charters/ippm-charter.html*.

[16] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. *FOCS*, 1989.

[17] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. *STOC*, 1989.

[18] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform. *Theor. Comput. Sci.*, 43(2-3):169–188, 1986.

[19] P. Laskowski and J. Chuang. Network monitors and contracting systems: competition and innovation. In *ACM SIGCOMM*, 2006.

[20] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. *SOSP*, 2003.

[21] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: detecting and isolating malicious routers. In *IEEE Internat. Conf. Dependable Systems and Networks*, 2005.

[22] M. Naor and G. N. Rothblum. Learning to impersonate. In *International Conf. on Machine Learning*, 2006.

[23] NIST. Hash function workshop. *http://csrc.nist.gov/pki/HashWorkshop/*.

[24] V. Padmanabhan and D. Simon. Secure traceroute to detect faulty or malicious routing. *HotNets-I*, 2002.

[25] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT, 1988.

[26] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Security and Privacy Symposium*, 2000.

[27] O. Reingold, L. Trevisan, and S. Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography Conference (TCC 2004)*, 2004.

[28] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving accuracy in end-to-end packet loss measurement. In *ACM SIGCOMM*, 2005.

[29] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and Whisper: Security mechanisms for BGP. In *USENIX NSDI*, 2004.