Untagging Tor: A Formal Treatment of Onion Encryption

Jean Paul Degabriele[1] and Martijn Stam[2]

[1] Department of Computer Science, TU Darmstadt, Germany.
jeanpaul.degabriele@cryptoplexity.de
[2] Department of Computer Science, University of Bristol, United Kingdom.
martijn.stam@bristol.ac.uk

Abstract. Tor is a primary tool for maintaining anonymity online. It provides a low-latency, circuit-based, bidirectional secure channel between two parties through a network of onion routers, with the aim of obscuring exactly who is talking to whom, even to adversaries controlling part of the network. Tor relies heavily on cryptographic techniques, yet its onion encryption scheme is susceptible to *tagging attacks* (Fu and Ling, 2009), which allow an active adversary controlling the first and last node of a circuit to deanonymize with near-certainty. This contrasts with less active traffic correlation attacks, where the same adversary can at best deanonymize with high probability. The Tor project has been actively looking to defend against tagging attacks and its most concrete alternative is proposal 261, which specifies a new onion encryption scheme based on a variable-input-length tweakable cipher.

We provide a formal treatment of low-latency, circuit-based onion encryption, relaxed to the unidirectional setting, by expanding existing secure channel notions to the new setting and introducing *circuit hiding* to capture the anonymity aspect of Tor. We demonstrate that circuit hiding prevents tagging attacks and show proposal 261's *relay* protocol is circuit hiding and thus resistant against tagging attacks.

Keywords: Anonymity · Onion Routing · Secure Channels · Tor · Tagging Attacks

1 Introduction

Anonymity as a separate security goal to confidentiality and integrity was recognized early on. Chaum [14] provided a number of suggestions for anonymous communication, of which his mix-nets later evolved into onion routing. Onion routing protocols come in a variety of flavours, depending on whether they are low-latency or not, whether they are circuit-oriented or ciphertext-oriented, the TCP/IP layer at which they operate, and a number of other factors. Examples include I2P [27], Mixminion [16], MorphMix [38] and Tarzan [23], but the best known and most widely used onion routing solution is Tor [20]. Tor is a low-latency, circuit-oriented onion routing protocol operating at the transport layer.

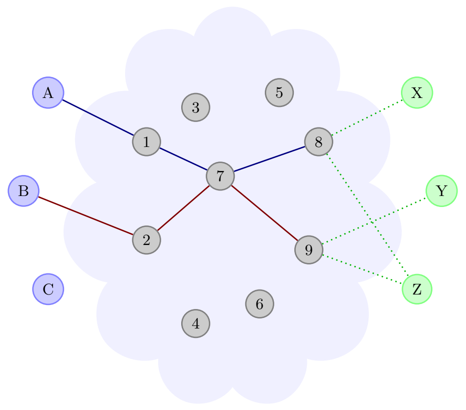

Fig. 1: Onion routing in a nutshell. Nodes A, B, and C are the onion proxies; nodes 1 to 9 are the onion routers making up the network; finally nodes X, Y, and Z are the destinations. User A created the dark blue circuit 1–7–8, using exit node 8 to communicate with destinations X and Z, whereas user B created the dark red circuit 2–7–9 and using its exit node 9 to communicate with destinations Y and Z.

Its original architecture was laid out in a quick succession of articles by Goldschlag, Reed, and Syverson [25,37,42]. The extent to which Tor and its brethren defend against (mostly) passive traffic correlation analysis has been an active research area [13,28,29,33,40], yet the impact of active attacks against the core cryptographic components on anonymity remains relatively unexplored. Indeed, it is not even clear what the formal design desiderata would be to provide any meaningful form of provable anonymity.

When talking about anonymity, it is worth bearing in mind the original goal set for Tor [25]: "The goal of onion routing is *not* to provide anonymous communication. Parties are free to (and usually should) identify themselves within a message. But the use of a public network should not automatically give away the identity and locations of the communication parties" (emphasis ours). In practice, an onion routing network (see Figure 1) enables two parties Anna (A) and Xavier (X) to route their communication through various intermediate nodes. As a result, there is no longer a direct link of communication between Anna and Xavier to observe and the hope is that their traffic gets lost in the masses. Ideally, even the intermediate nodes in direct contact with Anna and Xavier cannot link Anna and Xavier together.

The core components of Tor are the *link* protocol, the *circuit extend* protocol, the *relay* protocol, and the *stream* protocol. Any communication between any pair of interacting parties is secured by the link layer, which uses TLS, and all communication occurs on top of it. The circuit extend protocol establishes multi-hop tunnels called circuits between the sender and the receiver. In essence it uses public-key cryptography to exchange key material between nodes for onion encryption. The relay protocol is the component that actually handles the onion encryption and will be our main focus. The stream protocol operates over the relay protocol and is used to establish TCP connections, send data, etc.

At a very high level, the relay protocol operates as follows. The sender, which shares a symmetric key with every other node on the circuit, encrypts a message by applying multiple layers of encryption in succession, one for each node along the circuit. Specifically, a message is first encoded with a two-byte field of zeros and a SHA1 digest truncated to four bytes, and each layer of encryption then consists of 128-bit AES in counter mode. The resulting ciphertext, or *cell* in Tor's terminology, is then passed by the sender to the first node in the circuit. Each node in turn strips off one layer of encryption and either forwards the cell to the next node in the circuit or acts on that cell itself if it determines that it is the intended recipient. Note that only the final node in the circuit checks, and can check, the integrity by considering the redundancy introduced by the sender's encoding.

The combination of the final-node integrity check and the high level of malleability of counter-mode encryption leave Tor's relay protocol susceptible to the following *tagging attack* [24]. Assume an adversary controls the first and last nodes in a circuit. It can then 'tag' a cell c during the first hop by xoring it with some pattern δ, i.e. it sends $c \oplus \delta$ instead of c. If an honest exit node receives the corresponding cell, the integrity check will very likely fail, and the honest exit node will reject it. However, if the *adversary* controls the exit node, it can check for an invalid received cell c' that $c' \oplus \delta$ *does* pass the integrity check. Thus, the adversary has established that the two nodes are on the same circuit, and thereby linked the user (known to the first node) to its activities, as seen by the last node.

Superficially, tagging attacks expose a similar vulnerability as traffic correlation attacks by adversaries controlling both the first and last node of a circuit. Moreover, as Tor is a low-latency system, it cannot adequately protect against these passive traffic correlation attacks, which begs the question what active tagging attacks add to an adversary's arsenal. Indeed, back in 2004 when Tor was conceived [20], its authors already "accepted that our design is vulnerable to end-to-end timing attacks; so tagging attacks performed within the circuit provide no additional information to the attacker". Thus the choice for low latency—a compromise trading stronger security for usability—appeared to render tagging attacks redundant or even irrelevant, as seemingly equally powerful traffic correlation attacks are possible.

This perception changed around 2012, following an anonymous post on the Tor developers' mailing list by The23rd Raccoon, pointing out that tagging attacks *are* considerably more potent than traffic correlation attacks [36]: a successful tagging attack gives an adversary certainty when linking a circuit's entry and exit node, whereas for traffic correlation attacks a degree of uncertainty remains, including false positives where two nodes are incorrectly assumed to be on the same circuit. Consequently, tagging attacks scale better and with increased severity compared to traffic correlation attacks. We expand on these observations in Section 2.2, where we also address why detection (of the active tagging) does not lead to a satisfactory defense mechanism.

All in all, the Tor project has reversed its position and is currently seeking alternative onion encryption schemes that do protect against tagging attacks [31]. Whereas traffic correlation attacks cannot be prevented by cryptographic means (without sacrificing low latency), conceivably protection against tagging attacks without significant performance penalty is achievable.

Taking a broader perspective, we observe that while there has been ample work focusing on anonymity, circuit-based onion encryption *as a cryptographic primitive* has been largely overlooked. Yet in onion routing networks, anonymity is achieved through a combination of factors, such as the number of users in the system, the amount of traffic, and cryptographic mechanisms like onion encryption. The aforementioned tagging attacks against Tor clearly indicate that the latter is not well understood: it is unclear what properties the cryptographic component should provide, let alone how to do so.

Our contribution. Our aim is exactly this, to characterise what security properties *should* and *can* be expected from an onion encryption scheme. While we try to maintain as much generality as possible, our attention is focused on the setting typified by Tor. That is, we consider onion encryption for the case of low-latency and circuit-oriented systems, operating on top of a link protocol, like TLS, that secures communication between adjacent nodes. Tarzan and Morphmix, like Tor, fall within this category and are also captured by our models.

Our three design choices change the landscape quite significantly compared to high-latency mix-nets or public key ciphertext-oriented onion routing. In particular, requiring low-latency precludes the possibility of shuffling cells as in the case of mix-nets. A circuit-oriented architecture assumes the existence of a complementary protocol (in the case of Tor the circuit extend protocol) that sets up circuits across the network on which cells can be transmitted. Thus cells must follow predefined paths. In contrast, in a ciphertext-oriented architecture (as in I2P and Mixminion), each ciphertext can specify and follow a distinct path. This dichotomy corresponds quite closely to the distinction between symmetric and public-key onion encryption. One benefit of a circuit-oriented architecture is that, being stateful, it can protect against replay and reordering attacks. Finally, because the onion encryption operates on top of a link protocol, the adversary can only access the onion encryption if it controls some subset of the nodes in the network. We exploit this fact in our circuit hiding definition, but we don't make use of it in other security definitions as we can achieve the stronger definition with little effort or added complexity.

Clearly we would like an onion encryption scheme to provide confidentiality, integrity, and anonymity, ideally even if a subset of the nodes are under adversarial control. After establishing a syntax of circuit-based onion routing in Section 3, we adapt the end-to-end security notions for confidentiality and integrity of a secure unidirectional channel to the context of onion routing in Section 4, before tackling the most challenging and novel part, namely anonymity, in Section 5.

Anonymity in low-latency onion routing is never absolute and relies crucially on non-cryptographic factors—beyond what can be guaranteed by a suitable choice of onion encryption—such as the network size, its number of active users and the amount of traffic flowing through the network. Accordingly we do not aim for a full-blown definition of anonymity, but instead aim for a more refined security goal that can be achieved purely by cryptographic means (assuming ideal traffic conditions). Our proposed notion is that of circuit hiding, which roughly states that an adversary should not be able to learn any information about the circuits' topology in the network beyond what is inevitably leaked through the nodes that it controls. In particular this should hold even when the adversary is allowed to choose the messages that get encrypted and is able to re-order, inject, and manipulate ciphertexts on the network. Indeed the latter is exactly how tagging attacks operate, and consequently these are captured by our model.

Following on from the two potential directions to thwart tagging attacks [31], the only concrete proposal to date is Tor proposal 261 [32]. Our second contribution is a security analysis of the onion encryption scheme specified therein. Proposal 261 is based on AEZ [26] (see Section 6 for more details), but it could be instantiated with any other variable-input-length (VIL) tweakable cipher, such as Farfalle [9] or HHFHFH [8] which have both been suggested as alternatives to AEZ. Indeed, our analysis is general enough to apply to any such instantiation and thus its scope surpasses proposal 261.

Security in our framework guarantees protection from tagging attacks, but also ensures that it is not done at the detriment of some other security aspect. For instance, a naive solution to stop tagging attacks would be to extend Tor's counter mode AES with a MAC in an encrypt-then-MAC configuration. While this fix might foil tagging attacks, it would not suffice to guarantee circuit hiding, as the length of a circuit and a node's relative position within a circuit can now be inferred from the size of a cell. This is another instance of leakage on the circuits' topology that is captured by our notion.

We emphasize however that our analysis is limited in scope. In particular we only consider static node corruptions that are chosen by the adversary but fixed before it can interact with the network. Furthermore, we assume that circuits have already been established in a secure and anonymous way and we do not study how circuits should be chosen either.

Related work. Camenisch and Lysyanskaya [11] gave a formal security definition of public-key onion routing in the Universal Composability (UC) framework, as well as an alternative, compound game-based definition. For the latter they identify three core properties: correctness, integrity, and security. Combined with secure point-to-point channels these three imply the UC security notion. However their security model focuses on ciphertext-oriented architectures, where onion routers are stateless, and is therefore not applicable to Tor. For instance, their security definition does not and cannot capture circuits or provide protection against replay and reordering attacks.

Feigenbaum et al. [21] provide a black-box probabilistic analysis of onion routing based on a very high-level idealised functionality. Here an adversary is allowed to statically corrupt a fraction of the routers, so when a user selects a destination (for a circuit), the functionality randomly selects a path from the user to the destination and thereby determines whether the nodes adjacent to the user, resp. the destination, are corrupt or not. For corrupt adjacent nodes, the adversary will learn the user, resp. the destination, connected to it. In particular, if both adjacent nodes are corrupt, then the adversary will learn that a circuit has been established between the user and its destination. This model is useful for analysing the overall effect of traffic correlation attacks under the assumption that an adversary is capable to link traffic flowing in and out of the honest routers [43]. However, when considering how the cryptographic component affects this traffic linking assumption, their model is unsuitable.

Motivated by Tor, Backes et al. [3] propose another security definition for onion encryption in the UC framework. They consider a combined ideal functionality closely mirroring Tor's syntax, incorporating both the circuit establishment protocol and the onion encryption component. Meeting their security notion relies on the onion encryption being *predictably malleable*. Unfortunately, this predictable malleability is exactly what enables tagging attacks. In other words, schemes secure in their framework are *guaranteed* to be insecure against tagging attacks.

Danezis and Goldberg [17] propose Sphinx, a cryptographic packet format for relaying anonymized messages within high latency mix networks. It improves over prior constructions by being more space efficient, in part by replacing RSA encryption with elliptic curve cryptography. Sphinx is designed to protect against tagging attacks, but as it follows a ciphertext oriented architecture, is inapplicable to Tor.

In concurrent work, Rogaway and Zhang provide an independent treatment of onion encryption [39], see our full version [18] for a comparison.

2 Background and Preliminaries

2.1 An Overview of Tor

We now give a brief overview of how Tor works. A more detailed description can be found in Tor's introduction [20] or its protocol specification [19]; for a comparison of Tor with other anonymous communication systems see Danezis et al. [15].

In Tor, an end-to-end overlay network is formed wherein participating nodes, called onion routers (OR), relay messages across the overlay network. Users can run an onion proxy (OP) to access the Tor network (though we will use the terms sender, user, and onion proxy interchangeably). Onion routers maintain TLS connections with each other and onion proxies join the Tor network by establishing a TLS connection to one or more onion routers (the *link* protocol). All peer-to-peer communication occurs over these TLS connections. The aim of the network is to prevent outsiders or participating nodes from linking the recipient of a message to its source.

A user's application data is routed over fixed paths called *circuits*. A circuit is a path within the Tor network consisting of two or more ORs; the default is three. At the start of the circuit is an onion proxy which transmits data over the circuit, though Tor does not consider the OP to be part of the circuit. The exit node is the onion router responsible for delivering application data to the intended recipient (who may well reside outside the network). By default, the last node in a circuit acts as an exit node, but other nodes in the circuit may also act as exit nodes—a feature sometimes referred to as "leaky pipes". These variable exit nodes are supported in Tor by allowing multiple streams to run over the same circuit; for the most part we will ignore this complication and assume the exit node will be the last node of the circuit.

An onion proxy is responsible for establishing its circuits. To this end, it will select a sequence of onion routers in the circuit, where each node can appear only once. The proxy establishes a symmetric key with each of the routers in sequence using the *circuit extend* protocol: at each step the circuit is extended by one hop in a telescopic fashion, so the key agreement with the $i+1^{\text{th}}$ node in the circuit runs over the current, partial circuit with the i^{th} node temporarily taking on the role of exit node. Circuit establishment in Tor enables a bidirectional channel between the onion proxy and the exit node.

Once a circuit is established, the OP shares a symmetric key with each node in that circuit. Furthermore, each OR shares a distinct circuit identifier with each node that is adjacent to it in the circuit. The circuit is then used by the OP to instruct the exit node to establish a TCP connection to a specific address and port (the *stream* protocol). Data intended for this stream is then encapsulated in relay cells, and the *relay* protocol protects each cell with a checksum and multiple layers of encryption: the OP adds a layer of encryption (128-bit AES in counter mode) for each OR in the circuit. Upon receiving a relay cell, an OR looks up the cell's circuit identifier and uses the corresponding key to remove a layer of encryption. If the cell is headed away from the OP, the OR then checks whether the resulting cell has a valid checksum. The checksum is composed of two all-zero bytes and a four-byte digest computed through a seeded running hash over the data. If valid, the OR interprets the relay cell to be intended for itself (any node in the circuit can act as an exit node). Otherwise it looks up the circuit identifier and the OR for the next hop in the circuit, replaces the circuit identifier, and forwards the relay cell to the next OR in the circuit. If the OR at the end of the circuit received an unrecognised relay cell, an error has occurred and the circuit is torn down. An OP treats incoming relay cells similarly: it iteratively removes an encryption layer for each OR on the circuit from closest to farthest. If at any point the checksum is valid, the cell must have originated at the OR whose layer has just been removed.

In Tor, all data exchanged between nodes is encapsulated in *cells*. In the majority of cases, cells are of a fixed size. In version 4 and higher, fixed-size cells are 514 bytes long and consist of a header and a payload portion. The header is composed of a four-byte circuit identifier *id*, and a single-byte command field *cmd* indicating what to do with the cell's payload. Circuit identifiers are connection-specific, so as a cell travels along a circuit it will have a different circuit identifier on each OP/OR and OR/OR connection that it traverses. The cell's payload is protected using onion encryption, where each cell is additionally TLS encrypted on its OP/OR, resp. OR/OR connection. Based on their command field, cells are either control cells to be interpreted by the node that receives them, or relay cells which carry end-to-end stream data. Control cells serve to create, maintain and tear down circuits. Relay cells have an additional relay header located at the front of the payload composed of a two-byte stream identifier, a six-byte checksum, a two-byte length field, and a single-byte relay command field. The stream identifier allows multiple stream traffic to be multiplexed over the same circuit. The checksum is used by ORs to determine whether they are the intended recipient of the cell, while the length field specifies the size of the relay payload in bytes. Relay commands are exchanged between the Onion Proxy and the exit node to manage TCP streams, such as for instance to instruct an exit node to open a TCP connection to some destination specified in the relay payload.

Our focus is on the onion encryption component of Tor, which means we will abstract away most of the details of how an onion proxy initially connects to the Tor network, how it chooses which circuit to create, and how the telescopic key agreement operates. Instead, we will assume that all necessary keys have already been established in a secure manner and that secure channels between nodes (e.g. based on TLS) are readily available. We collapse the *stream* and the *relay* protocols and only directly consider sending arbitrary length messages, thus ignoring complications arising from treating data as a stream [22].

2.2 On The Relative Severity of Tagging Attacks

We have already alluded to the similarity between tagging attacks and traffic correlation attacks, which in turn raises the question as to why should we bother with tagging attacks at all when seemingly equally powerful attacks are possible. There is indication however that tagging attacks can be significantly more damaging than traffic correlation attacks. The arguments in support of this claim stem from the analysis in two anonymous posts on the Tor developers' mailing list by The23rd Raccoon from 2008 and 2012 [35,36]. In turn these observations prompted the Tor project to reverse its decision and seek to protect against tagging attacks [31, 32]. We here attempt to give some insight into this rationale but refer the reader to the actual posts for further details.

The main distinctive advantage of tagging attacks over traffic correlation attacks is that a circuit can be confirmed with a zero chance of a false positive (i.e. two end points being categorised as belonging to the same circuit when in reality they do not). In contrast, traffic correlation techniques inevitably incur false positives with non-zero probability. Moroever, the base rate fallacy implies that even a relatively small false positive rate severely reduces the overall efficacy of traffic correlation attacks as the network size increases [35]. While the original post [35] did not make any mention of tagging attacks it is easy to see that tagging attacks are immune to this phenomenon and therefore scale much better than traffic correlation attacks.

Another argument in support of the severity of tagging attacks is their inherent "amplification" effect as described by The23rd Raccoon in 2012 [36]. (Perhaps amplification is not the most appropriate term but to avoid confusion we stick with The23rd Raccoon's choice.) The amplification relies on the tear down of circuits as soon as a tagged cell is not untagged at the exit of the network (and similarly, whenever cells that were not previously tagged are "untagged"). The immediate effect is that uncompromised circuits will be automatically filtered out and the adversary does not have to dedicate further resources to them. A secondary effect is that, when the OP attempts to re-establish a circuit using a new path, with some probability both entry and exit routers will be under adversarial control. Thus tagging attacks bias the creation of more compromised circuits. In principle, this amplification could be simulated using a traffic correlation attack by actively tearing down uncorrelated circuits, though again, false positives limit the efficacy of this approach [36].

Regrettably, tagging attacks cannot easily be prevented by detection and subsequent eviction of dishonest routers. For instance, although tagging attacks have been known since at least 2004, in 2014 they were successfully deployed against Tor without being noticed until months later [2]. Secondly, a client can only detect modification as a circuit failure but the natural failure rate in the Tor network is high enough to complicate timely detection of an attack. Moreover, even if a circuit failure is correctly classified as an attack, identifying the malicious onion routers is far from obvious. It requires independent onion routers to collaborate, including a mechanism to resolve disputes as misbehaving routers could manipulate the evidence in order to shift the blame on other routers. Such collaboration is further hampered as the required exchange of information should not allow the reconstruction of the affected circuits as it would de-anonymise their users in the process—precisely what we are trying to prevent in the first place. Finally, an attacker in full control of the exit nodes through which the tagged traffic flows can avoid detection altogether. Using tagging attack in conjunction with preliminary traffic analysis could realistically lead to such a scenario.

2.3 Notation

If $\mathcal{S}$ is a finite set then $|\mathcal{S}|$ denotes its size, and $y \leftarrow_{\$} \mathcal{S}$ denotes the process of selecting an element from $\mathcal{S}$ uniformly at random and assigning it to y. An oracle may return the special symbol $\lightning$ to suppress output; in contrast $\perp$ denotes an error message that is output by some scheme.

We denote vectors in bold letters or explicitly by listing their components in between $[\,]$. For any vector $\mathbf{v}$, we denote its i^{th} component by $\mathbf{v}[i]$, its size by $|\mathbf{v}|$ and we endow it with a function $\mathbf{v}.\mathsf{append}(e)$ that extends $\mathbf{v}$ with a new component of value e. We use $[e]_1^n$ to denote a vector of size n whose entries are all set to e. We also make use of queue structures, where for any queue Q

the function calls $Q.\mathsf{enqueue}()$ and $Q.\mathsf{dequeue}()$ bear their usual meaning. Unless otherwise specified, all vectors and queues are initially empty.

3 Modelling Onion Routing Networks

Ultimately, our goal is to quantify how well the cryptologic component of cell creation and processing provides security and anonymity, even against adaptive adversaries. Our model abstracts away certain aspects that are highly relevant in practice (e.g. key management and traffic analysis), but that are to a large extent orthogonal to the cryptographic channel security. To ensure our formalism reasonably matches intuition, we embedded some Tor specific design choices into our syntax, yet our syntax is considerably more general in order to capture alternative cryptographic solutions as well. As a result, we strike a balance to avoid needless complexity as much as possible.

We consider two types of roles, corresponding to Tor's onion proxies and onion routers, respectively. The onion routers and proxies are modelled by nodes in a graph, with the (directed) edges representing possible direct communication. Our assumption is that each directed edge corresponds to an independent, unidirectional secure channel, and that the graph is a complete directed graph, allowing any party to communicate securely and directly (but not anonymously!) with any other party. If desired, one could consider other graphs to represent topological restrictions. All parties have a unique, publicly known identifier, and can take on the role of both proxy and router—even though we often write as if these are completely different entities.

As in Tor, the onion proxies are responsible for initializing circuits and for encrypting messages to a circuit, which we assume is used for *unidirectional* anonymous communication. A circuit consists of an onion proxy and a path $\mathbf{p}$ through the graph of onion routers. The path should be acyclic and its length is denoted ℓ. The circuit is then represented as a vector of nodes $\mathbf{p}[1], \ldots, \mathbf{p}[\ell]$ where we abuse $\mathbf{p}[0]$ to refer to the circuit's onion proxy so $\mathbf{p}$ allows the identification of not only the ℓ routers, but also the proxy. By convention, we set $\mathbf{p}[\ell+1] = \oslash$ to indicate the end of a circuit, where the symbol $\oslash$ is reserved solely for this purpose and cannot be assigned to any node. For any circuit, we use the terms sending node, receiving node and forwarding nodes to refer respectively to the OP, the path's last node $\mathbf{p}[\ell]$ and intermediate nodes $\mathbf{p}[1], \ldots, \mathbf{p}[\ell-1]$ in the path. Note that our receiving node corresponds to Tor's exit node, whereas the party outside the network with which the exit node communicates is beyond the scope of our formalism.

Both onion proxies and onion routers maintain a vector of states, containing a state for each of the circuits they are involved with. There is a notable difference in their use, as proxies use their state for encryption and will know which circuit it is for (and therefore which 'state' to use), whereas for routers, upon receipt of a cell they will first have to figure out which of its circuits it is intended for, if any. Accordingly, we split decryption into two separate stages, D and $\bar{\mathsf{D}}$, where D is responsible for figuring out the relevant circuit and $\bar{\mathsf{D}}$ for the proper processing

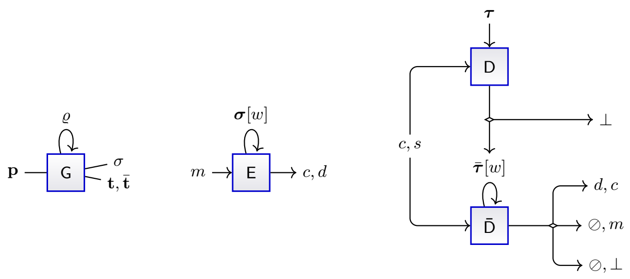

Fig. 2: Our syntax illustrated, showing the various possible outcomes during decryption. The end-of-circuit symbol $\oslash$ indicates that the current node is the intended recipient; the loops above algorithms indicate a state update.

of the cell. Note that the identity of the cell's sender can be used to help identify the circuit, though not necessarily uniquely as multiple circuits could be routed along the same edge.

We require that a node maintains its individual decryption states in two separate state vectors $\boldsymbol{\tau}$ and $\bar{\boldsymbol{\tau}}$, relevant for D and $\bar{\mathsf{D}}$, respectively. Each time a new cicuit is created, every forwarding and receiving node in that path will append a new component to its decryption state vectors $\boldsymbol{\tau}$ and $\bar{\boldsymbol{\tau}}$.

Our two-stage model $(\mathsf{D}, \bar{\mathsf{D}})$ for the processing of cells, is very much a choice for which level of generality to strive for. On the one hand, it reflects practical protocol designs such as Tor, without being overly prescriptive on quite how routing has to work. On the other hand, our model is evidently less general than a single stage model, that might allow arbitrary changes to its state. Our split in two stages, coupled with the restrictions on how the state looks and can be affected, guarantees that the processing of cells for one circuit cannot unduly influence the later processing of a cell associated to a different circuit. This guaranteed *robustness* significantly simplifies the definition of security games later on, as all circuits which the adversary has not interacted with, will still behave correctly. For a more general syntax, robustness does not follow automatically and would have to be modelled separately.

3.1 Onion Encryption

A (symmetric) *onion encryption scheme* $\mathsf{OE} = (\mathsf{G}, \mathsf{E}, \mathsf{D}, \bar{\mathsf{D}})$ is a quadruple of algorithms (see Fig. 2) to which we associate a message space $\mathsf{MsgSp} \subseteq \{0,1\}^*$ and a cell space $\mathsf{CelSp} \subseteq \{0,1\}^*$.

– The stateful circuit creation algorithm G is an abstraction of how circuits are created (which in reality is more likely to be an interactive process). It takes as input a path $\mathbf{p}$ that is not allowed to loop (by using the same router multiple times) and includes the proxy $\mathbf{p}[0]$. It updates its own state ϱ (initially

$\varrho = \varepsilon$) and returns an initial encryption state σ (given to $\mathbf{p}[0]$) and two vectors, $\mathbf{t}$ and $\bar{\mathbf{t}}$, of initial decryption state components, one for each router in the path, so $|\mathbf{t}| = |\bar{\mathbf{t}}| = |\mathbf{p}|$. Upon receipt of their respective entries of $\mathbf{t}$ and $\bar{\mathbf{t}}$, the routers append these entries to their decryption state pair $(\boldsymbol{\tau}, \bar{\boldsymbol{\tau}})$. That is, if $\mathbf{p} = [a, b, c, d, e]$, we update the individual decryption states by the following sequence of operations: $\boldsymbol{\tau}_b.\mathsf{append}(\mathbf{t}[1]), \bar{\boldsymbol{\tau}}_b.\mathsf{append}(\bar{\mathbf{t}}[1]), \ldots, \boldsymbol{\tau}_e.\mathsf{append}(\mathbf{t}[4]), \bar{\boldsymbol{\tau}}_e.\mathsf{append}(\bar{\mathbf{t}}[4])$. Similarly the proxy's encryption state vector is updated by $\boldsymbol{\sigma}_a.\mathsf{append}(\sigma)$. As shown above, we will indicate the identity of the node to which a state vector or state variable belongs through its subscript. See also $\textsc{Add}(\mathbf{p})$ (Fig. 3) for G in action.

- The algorithm E is used by a proxy to send messages to one of its circuits. Given the current encryption state $\boldsymbol{\sigma}[w]$ for a circuit indexed locally by w and a message $m \in \mathsf{MsgSp}$, the algorithm E updates the encryption state and returns an initial cell $c \in \mathsf{CelSp}$ as well as the identity d of the router to which the cell has to be forwarded to.

- The deterministic algorithms D and $\bar{\mathsf{D}}$ are jointly responsible for processing an incoming cell c by a router. In the first stage, D associates the cell c to one of its circuits, where it can also use the identity of the source node s from which it received the cell. Importantly, D takes as additional input the node's entire first decryption state $\boldsymbol{\tau}$, but *without* the possibility to change this state. It returns a 'local' index w indicating to which circuit it has associated the cell and hence which component of the second decryption state $\bar{\boldsymbol{\tau}}$ should be used by $\bar{\mathsf{D}}$ to process the cell. The symbol $\perp$ indicates that the cell could not be associated to a circuit.
  In the second stage, $\bar{\mathsf{D}}$ takes the state component $\bar{\boldsymbol{\tau}}[w]$, as well as the source node and cell. It can update the decryption state component (though not any other part of the state) and return an output string x and a destination node d. The value d indicates the node to which the string x is to be forwarded, where $x \in \mathsf{CelSp}$. Alternatively, if $d = \oslash$, the router knows it is the intended recipient, in which case $x \in \mathsf{MsgSp} \cup \{\perp\}$.

A cell's trajectory and lifecycle. Once E has output a cell c and an initial router d, we could start following that cell through the network: present the cell c to d, receiving new cell c' and destination d', so forward c' to d', etc. until a router either outputs $\perp$ or $d' = \oslash$. This process determines the *trajectory* of the cell, namely the chronological sequence of routers that process it, as well as the *lifecycle* of the cell, namely the sequence of cells that is input to routers during this processing.

In the description above, we implicitly assumed that the routers on the cell's trajectory were exclusively processing the cells in its lifecycle. In reality there will be much more traffic that the routers will process. This additional processing can affect the routers' states and consequently change the real-life trajectory and lifecycle of a cell. Our syntactical choices, such as deterministic processing by $\bar{\mathsf{D}}$, ensure that the lifecycle of a cell is fixed, as long as the real-life trajectory matches the path corresponding to the cell's intended circuit (cf. the security notion trajectory integrity, see the full version [18]). The ability to effectively predetermine a cell's lifecycle will turn out crucial when defining the security notion circuit hiding (Definition 4); it was exploited using a slightly different formalism in the context of public key, circuitless onion routing [11].

Local versus global perspective. A key goal of onion routing is to ensure that routers are unable to link the recipient of a message to the proxy from which it originated, unless all the routers on a circuit collude. This necessitates that the router's view of a circuit is local: it knows which of its own circuits a cell belongs to (D's output), but otherwise a router should only be aware of the nodes that are directly adjacent to it.

Yet, when formalizing security notions (or correctness), we will need a global view and a way to move effortlessly from a router's local perspective to a more global view. To this end, we associate a global circuit index to each circuit upon creation and define the function map that takes a global circuit index and router index on the corresponding circuit and maps it to the node identifier and local circuit index. We allow the router index to be 0, so for instance $(v, w) = \mathsf{map}(i, 0)$ indicates that $v = \mathbf{p}_i[0]$ is the proxy for the circuit with global index i. The partial inverse map map^{-1} takes a node's identifier (which cannot be $\oslash$) and its local circuit index, and maps it back to the global view: which circuit is this and how far along the circuit does v occur. Both map and map^{-1} are dynamically defined (as new circuits can be created) and both are only ever called on their proper domains (values for which the functions are by then well-defined), with the convention that $\mathsf{map}(i, |\mathbf{p}_i| + 1)$ and $\mathsf{map}(i, -1)$ are set to $(\oslash, 0)$.

3.2 Correctness

Correctness guarantees that honestly generated cells are routed correctly and decrypt to the original messages at their intended destination in the same order as they were sent. Correctness should hold regardless of which circuits are created when, or the order in which cells are processed, as long as the order of cells belonging to the same circuit is preserved.

We formulate correctness through a game (see Fig. 3) whereby a scheduler is allowed to create circuits, choose the messages to be sent by the sending nodes, and determine the order in which individual routers process cells across different circuits. Reordering of cells belonging to different circuits models unpredictability of delays across the physical network, as well as the router's (limited) liberty to mix up the processing of cells to hamper traffic analysis. The scheduler however is *not* allowed to tamper with cells.

Concretely, for each circuit i we maintain a list $\mathbf{m}_i$ of the messages being sent through that circuit (using ENC), and check at the router's end (PASS) whether the messages arrive in order (the counter ctr_i indicates the next message on $\mathbf{m}_i$ that should be received). Moreover, for each circuit i and each of the $|\mathbf{p}_i|$ routers on its path (counted using j), we maintain a queue Q^i_j to keep track of the cells that are waiting to be processed by that router. Thus processing a cell by a forwarding router results in dequeuing for the current router and circuit, and enqueuing for its successor router.

Definition 1 (Correctness). *An onion encryption scheme* OE *is said to be correct if for all scheduling algorithms* $\mathcal{S}$ *(including computationally unbounded ones) it holds that:*

$$\Pr\left[\,\mathrm{TRANSMIT}^{\mathcal{S}}_{\mathsf{OE}} \Rightarrow \mathsf{true}\,\right] = 0\ ,$$

where the game TRANSMIT *is given in Figure 3.*

```
Game $\mathrm{TRANSMIT}^{\mathcal{S}}_{\mathsf{OE}}$
  $\varrho \leftarrow \varepsilon$; $n \leftarrow 0$
  $\mathsf{win} \leftarrow \mathsf{false}$
  $\mathcal{S}^{\textsc{Add},\textsc{Enc},\textsc{Pass}}$
  return $\mathsf{win}$

$\textsc{Pass}(i, j)$
  if $\neg(0 < j \le \ell_i) \vee Q^i_j = [\,]$
    return $\lightning$
  $c \leftarrow Q^i_j.\mathsf{dequeue}()$
  $s \leftarrow \mathbf{p}_i[j-1]$
  $(v, w') \leftarrow \mathsf{map}(i, j)$
  $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
  if $w \neq w'$
    $\mathsf{win} \leftarrow \mathsf{true}$
    return $\perp$
  $(\bar{\boldsymbol{\tau}}_v[w], d, x) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
  if $j < \ell_i \wedge d = \mathbf{p}_i[j+1]$
    $Q^i_{j+1}.\mathsf{enqueue}(x)$
  elseif $j = \ell_i \wedge d = \oslash \wedge x = \mathbf{m}_i[\mathsf{ctr}_i]$
    $\mathsf{ctr}_i \leftarrow \mathsf{ctr}_i + 1$
  else $\mathsf{win} \leftarrow \mathsf{true}$
  return $(d, x)$

$\textsc{Enc}(i, m)$
  $(v, w) \leftarrow \mathsf{map}(i, 0)$
  $\mathbf{m}_i.\mathsf{append}(m)$
  $(\boldsymbol{\sigma}_v[w], d, c) \leftarrow \mathsf{E}(\boldsymbol{\sigma}_v[w], m)$
  if $d \neq \mathbf{p}_i[1]$
    $\mathsf{win} \leftarrow \mathsf{true}$
  else
    $Q^i_1.\mathsf{enqueue}(c)$
  return $(d, c)$

$\textsc{Add}(\mathbf{p})$
  if $|\mathbf{p}| \ge 1$
    $n \leftarrow n + 1$
    $\mathbf{p}_n \leftarrow \mathbf{p}$; $\ell_n \leftarrow |\mathbf{p}|$
    $\mathsf{ctr}_n \leftarrow 1$
    $(\varrho, \sigma, \mathbf{t}, \bar{\mathbf{t}}) \leftarrow \mathsf{G}(\varrho, \mathbf{p})$
    if $|\mathbf{t}| \neq \ell_n \vee |\bar{\mathbf{t}}| \neq \ell_n$
      $\mathsf{win} \leftarrow \mathsf{true}$
    $\boldsymbol{\sigma}_{\mathbf{p}[0]}.\mathsf{append}(\sigma)$
    for $j = 1$ to $\ell_n$
      $v \leftarrow \mathbf{p}[j]$
      $\boldsymbol{\tau}_v.\mathsf{append}(\mathbf{t}[j])$
      $\bar{\boldsymbol{\tau}}_v.\mathsf{append}(\bar{\mathbf{t}}[j])$
  return $n$
```

Fig. 3: The TRANSMIT game used to define correctness for onion encryption schemes.

3.3 Security

Onion routing networks should satisfy a range of security notions. In Section 5 we will deal with anonymity (in the form of circuit-hiding), and in Section 4 we concentrate on integrity and confidentiality, where the goal is that every circuit should implement a secure channel, even if the adversary has full control of the intermediate routers. Though we cannot give full control to an adversary in the anonymity setting, our security definitions in both sections share a number of modelling choices, as explained below.

Firstly, all our security notions are game-based where we simply define an adversary's advantage, without making an explicit and precise statement of what constitutes "secure". This concrete security approach is gaining traction for real world cryptosystems and would be harder to achieve in for instance an asymptotic UC framework. Secondly, all our formal definitions are multi-user definitions in the context of the entire routing network. For simplicity and wherever possible, our intuitive explanations only address what happens for a single circuit.

The customary threat model is to protect against adversaries who can "observe some fraction of network traffic; who can generate, modify, delete, or delay traffic; who can operate onion routers of their own; and who can compromise some fraction of the onion routers." [20]. When we map this threat model to our formal model, we first need to factor in the effect of the secure, unidirectional node-to-node communication. On a single edge, a passive outside adversary will be able to see the timing and volume of traffic. While this is extremely potent information to perform traffic analysis, our focus on the core cryptographic component renders this information largely out of scope. An active outside adversary can delay traffic on an edge (or delete all future traffic), but the edge's channel security will prevent it from inserting, modifying (including reordering and replaying), or deleting any of these cells. However, if a router is set to receive two cells from different routers, the adversary could control which one will arrive first. Fortunately, our two-stage approach to decryption with the router's state update restricted to a single circuit, makes the order in which cells associated to different circuits are processed irrelevant. Thus, for circuit hiding (Section 5) we restrict the adversary to the network interface it obtains from the compromised onion routers. For channel security we expand the adversary's power slightly (see Section 4).

The operation and compromise of routers is modelled by *selective* corruptions, where the adversary has to specify the set $\mathcal{C}$ of nodes it wishes to corrupt at the outset. For corrupted nodes, an adversary will learn the state of the router (incl. future updates), have access to all incoming cells to that router, and have full control over the cells being sent out to other routers. Recalling that circuit creation G outputs the triple $(\sigma, \mathbf{t}, \bar{\mathbf{t}})$ encoding the state updates of the proxy and routers on the circuit's path, we denote with $(\sigma, \mathbf{t}, \bar{\mathbf{t}})|_{\mathcal{C}}$ those state updates that are associated with corrupted nodes.

Our choice for selective corruptions only is informed by the often unforeseen complications that adaptive corruptions bring with them (see for instance selective opening attacks [4] and non-committing encryption [34]). Moreover, formalizing secure channels in a multi-user setting is relatively uncommon (cf. [30]) and introducing adaptive corruptions is, as far as we are aware of, unexplored.

4 Channel Security

We model channel security by considering both integrity and confidentiality, where we concentrate on the end-to-end effect (so plaintext integrity instead of ciphertext integrity and left-or-right indistinguishability instead of ciphertext indistinguishability). Moreover, we consider a slightly stronger threat model as the one alluded to above: even if an adversary has not corrupted a node, we will allow the adversary full control over its incoming and outgoing edges. Thus the end-to-end channel security of a circuit established by Tor should rely purely on its two end points not being compromised. Consequently, the unidirectional node-to-node security provided by TLS is of no use to establish channel security.

Plaintext integrity. Plaintext integrity guarantees that, even in the presence of an adversary with *almost* full knowledge of all states and full control of the network, an honest receiving node can be reassured that the messages it outputs correspond to those being sent (assuming the sending node is uncompromised). This captures the inability for an adversary to inject, modify, reorder, or replay messages.

The game PINT (Figure 4) models plaintext integrity. For each circuit with an honest proxy, we maintain a list $\mathbf{m}_i$ to check whether messages arrive unmodified and in the correct order at the honest receiver. The oracle $\textsc{Proc}(s, v, c)$ models node v's processing of cell c received from s. We do not insist that s is corrupt, thus we allow an adversary to inject cells even on edges for which it does not control the sending node, notwithstanding our assumption on secure node-to-node communication. This modest strengthening of the notion results in a slightly cleaner game.

Mirroring the correctness game, the counter ctr_n indicates the next message on $\mathbf{m}_n$ that should be received. As one would when defining plaintext integrity for ordinary channels, if the honest receiver accepts a message that wasn't sent (in that order), the adversary wins. Additionally—and this concept appears unique for routing networks—if an intermediate forwarding router believes a cell contains a valid message intended for it, the adversary wins. This win is a consequence of our choice *not* to allow "leaky pipes" [20].

Also note that indexing a vector component that does not exist is assumed to return a special symbol outside the set $\{0,1\}^*$. That is, for any vector $\mathbf{m}$ and any $k > 0$, if $|\mathbf{m}| = t$ then $\mathbf{m}[t+k] \neq \varepsilon$. In particular if the onion encryption scheme allows 'dummy' cells that decrypt to the empty string, an adversary that is able to forge such a dummy cell is deemed successful in the game PINT.

Definition 2. *The plaintext integrity advantage of adversary* $\mathcal{A}$ *against* OE *is defined by*

$$\mathbf{Adv}^{\mathrm{PINT}}_{\mathsf{OE}}(\mathcal{A}) = \Pr\left[\, \mathrm{PINT}^{\mathcal{A}}_{\mathsf{OE}} \Rightarrow \mathsf{true} \,\right] ,$$

where the game PINT *is given in Figure 4.*

```
Game $\mathrm{PINT}^{\mathcal{A}}_{\mathsf{OE}}$
    $\varrho \leftarrow \varepsilon$; $n \leftarrow 0$
    $\mathsf{win} \leftarrow \mathsf{false}$
    $(\mathcal{C}, \mathsf{st}) \leftarrow \mathcal{A}_1$
    $\mathcal{A}_2^{\textsc{Add},\textsc{Enc},\textsc{Proc}}(\mathsf{st})$
    return $\mathsf{win}$
```

```
$\textsc{Enc}(i, m)$
    $(v, w) \leftarrow \mathsf{map}(i, 0)$
    if $v \in \mathcal{C}$
        return $\lightning$
    $\mathbf{m}_i.\mathsf{append}(m)$
    $(\boldsymbol{\sigma}_v[w], d, c) \leftarrow \mathsf{E}(\boldsymbol{\sigma}_v[w], m)$
    return $(d, c)$
```

```
$\textsc{Add}(\mathbf{p})$
    if $|\mathbf{p}| \geq 1$
        $n \leftarrow n + 1$
        $\mathbf{p}_n \leftarrow \mathbf{p}$; $\ell_n \leftarrow |\mathbf{p}|$
        $\mathsf{ctr}_n \leftarrow 1$; $\mathsf{sync}_n \leftarrow \mathsf{true}$
        $(\varrho, \sigma, \mathbf{t}, \bar{\mathbf{t}}) \leftarrow \mathsf{G}(\varrho, \mathbf{p})$
        $\boldsymbol{\sigma}_{\mathbf{p}[0]}.\mathsf{append}(\sigma)$
        for $j = 1$ to $\ell_n$
            $v \leftarrow \mathbf{p}[j]$
            $\boldsymbol{\tau}_v.\mathsf{append}(\mathbf{t}[j])$
            $\bar{\boldsymbol{\tau}}_v.\mathsf{append}(\bar{\mathbf{t}}[j])$
    return $(\sigma, \mathbf{t}, \bar{\mathbf{t}})|_{\mathcal{C}}$
```

```
$\textsc{Proc}(s, v, c)$
    if $v \in \mathcal{C}$
        return $\lightning$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    if $w = \perp$
        return $\perp$
    $(i, j) \leftarrow \mathsf{map}^{-1}(v, w)$
    $(\bar{\boldsymbol{\tau}}_v[w], d, x) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
    if $d = \oslash \wedge x \neq \perp$
        if $j = \ell_i \wedge x = \mathbf{m}_i[\mathsf{ctr}_i]$
            $\mathsf{ctr}_i \leftarrow \mathsf{ctr}_i + 1$
        else
            $\mathsf{win} \leftarrow \mathsf{true}$
    return $(d, x)$
```

Fig. 4: The PINT game used to define plaintext integrity for onion encryption schemes. The sync_n flags in the $\textsc{Add}(\mathbf{p})$ oracle are used in later games.

Confidentiality. Confidentiality guarantees that an adversary gains no knowledge about the content of messages being sent on a circuit, as long as both the receiving and sending nodes are uncompromised. Otherwise, the adversary may have full knowledge of all states (except ϱ) and full control of the network. As usual for confidentiality, it is possible to provide both a passive 'CPA' and an active 'CCA' variant. Our game (Figure 5) captures 'chosen cell attacks'; for the weaker chosen plaintext attack variant simply remove adversarial access to $\textsc{Proc}$.

The mechanism to define chosen cell attacks is an adaptation of left-or-right CCA indistinguishability for stateful encryption [6], combined with the plaintext-oriented suppression of 'decryption' queries as introduced in the context of RCCA security [12]. (The 'R' from RCCA for replayable has become a misnomer in our context.) In our view, a plaintext-oriented CCA notion better matches the philosophy of end-to-end security (like plaintext integrity), making it cleaner to define and less dependent on assumptions how the channel is implemented. We opted for left-or-right over real-or-random as the former in general appears slightly more robust in a multi-user setting [7], despite their fairly tight equivalence for a single-instance.

The lists $\mathbf{m}_i$ are as before, though as we are considering a left-or-right notion, they will either contain all the 'left' or all the 'right' messages. The Proc oracle plays the role of decryption oracle, which would result in trivial wins if an adversary were allowed to learn the decryption result of the final cell. As long as the receiving router is in-sync with the proxy, the message to be output will be suppressed from the adversary (by returning $\lightning$ instead). Once a single message deviates (which includes the error symbol $\perp$) the receiving node is deemed out-of-sync and henceforth its output will no longer be suppressed. Intermediate nodes are deemed out-of-sync from the get go, so their output will never be suppressed; of course an adversary might well have corrupted all forwarding nodes in the circuit.

Definition 3. *The plaintext confidentiality advantage of adversary $\mathcal{A}$ against* OE *is defined by*

$$\mathbf{Adv}^{\mathrm{LOR}}_{\mathsf{OE}}(\mathcal{A}) = 2 \cdot \Pr\left[\, \mathrm{LOR}^{\mathcal{A}}_{\mathsf{OE}} \Rightarrow \mathsf{true} \,\right] - 1 \ ,$$

where the game LOR *is given in Figure 5.*

Trajectory integrity. If all behaviour is honest, a cell is guaranteed by correctness to follow its intended trajectory. However, an adversary could interfere by injecting and modifying traffic potentially affecting the routing of cells. We provide a formal definition of inconsistent routing in the full version [18] under the name (cell) trajectory integrity. We assume an adversary has full control over the network and it knows all the parties' secrets, notwithstanding we assume all parties will still honestly process cells using the $\textsc{Proc}(s, v, c)$ oracle. Both D and $\bar{\mathsf{D}}$ could lead to inconsistent routing, where a cell's trajectory does not match the path of a single circuit, or does not match the path of the circuit originally used by the proxy to create the cell (modelled by the $\textsc{Enc}(i, m)$ oracle).

5 Anonymity

Overview. In an onion routing network, anonymity relies on a number of factors, such as the size of the network, the amount of traffic, the length of the anonymous channels (circuits), etc. Anonymity services such as sender anonymity and unlinkability can only be attained if the topology of the network of circuits remains hidden. We investigate how the *cryptographic* properties of an onion encryption scheme can contribute towards hiding the network's topology from an adversary.

```
Game $\mathrm{LOR}^{\mathcal{A}}_{\mathsf{OE}}$
    $\varrho \leftarrow \varepsilon$; $n \leftarrow 0$
    $\mathsf{win} \leftarrow \mathsf{false}$
    $b \leftarrow_{\$} \{0,1\}$
    $(\mathcal{C}, \mathsf{st}) \leftarrow \mathcal{A}_1$
    $b' \leftarrow \mathcal{A}_2^{\textsc{Add},\textsc{Enc},\textsc{Proc}}(\mathsf{st})$
    return $b = b'$

$\textsc{Enc}(i, m_0, m_1)$
    $(v, w) \leftarrow \mathsf{map}(i, 0)$
    if $v \in \mathcal{C} \vee \mathbf{p}_i[\ell_i] \in \mathcal{C} \vee |m_0| \neq |m_1|$
        return $\lightning$
    $\mathbf{m}_i.\mathsf{append}(m_b)$
    $(\boldsymbol{\sigma}_v[w], d, c) \leftarrow \mathsf{E}(\boldsymbol{\sigma}_v[w], m_b)$
    return $(d, c)$

$\textsc{Proc}(s, v, c)$
    if $v \in \mathcal{C}$
        return $\lightning$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    if $w = \perp$
        return $\perp$
    $(i, j) \leftarrow \mathsf{map}^{-1}(v, w)$
    $(\bar{\boldsymbol{\tau}}_v[w], d, x) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
    if $j = \ell_i \wedge d = \oslash$
        if $c = \mathbf{m}_i[\mathsf{ctr}_i] \wedge \mathsf{sync}_i = \mathsf{true}$
            $\mathsf{ctr}_i \leftarrow \mathsf{ctr}_i + 1$
            return $\lightning$
        else
            $\mathsf{sync}_i \leftarrow \mathsf{false}$
    return $(d, x)$
```

Fig. 5: The LOR game used to define left-or-right indistinguishability for onion encryption schemes. For the $\textsc{Add}(\mathbf{p})$ oracle refer to Figure 4.

Our starting point is an indistinguishability game where the adversary gets to interact with one of two possible networks of his choice, and is required to guess which network it is interacting with. The adversary's interaction with and view of the network is facilitated by the nodes of the network it has corrupted and thus controls. An adversary controlling part of the network will inevitably gain partial information about that network, in particular about the topology of its circuits. For instance, for a corrupted node, an adversary will always be able to learn the previous and subsequent nodes of each of the corrupted node's circuits. For a circuit being routed through a contiguous sequence of corrupted nodes, the adversary can piece together the *directed subcircuit* as formed by those corrupted nodes and their adjacent honest nodes. The restrictions on an adversary's behaviour to avoid 'trivial' wins (e.g. if these observable, directed subcircuits differ between the two worlds) form a critical component of our circuit hiding game C-HIDE.

In the first stage of this game (see Figure 7), the adversary $\mathcal{A}_1$ specifies a pair of vectors of circuits $\mathcal{W}_0$ and $\mathcal{W}_1$, and a set of corrupted nodes $\mathcal{C}$. Subject to a number of checks to avoid trivial wins (implemented by the predicate $\textsc{Valid}$ as explained below), the game uses the procedure $\textsc{Init-Circ}$ to initialize either the $\mathcal{W}_0$ or $\mathcal{W}_1$ network. The adversary is given $\tau_{\mathcal{C}}$ containing the states of corrupted nodes, but with a twist: after all circuits have been created by $\textsc{Init-Circ}$, the router's decryption-state vectors are all shuffled (and the map function will refer to the state post-shuffle). The shuffling reflects a secure implementation that avoids "order" correlation attacks by linking traffic through the order in which circuits were set up. For further justification on why this shuffling is necessary, we refer the reader to the full version of this paper [18]. Without this shuffling, security against active attacks appears a lot harder to achieve in the absence of very strong cell integrity.

In addition, the adversary is given access to the network by means of two oracles. The encryption oracle ENC can be used to trigger honest proxies to encrypt any message for one of its circuits. The network oracle NET provides collective and suitably restricted access to the honest routers in the network, as explained below. The goal of the adversary it to guess which of the two networks ($\mathcal{W}_0$ or $\mathcal{W}_1$) it is interacting with.

Below we will often refer to *segments* of a circuit. A circuit segment is defined to be a maximal subpath of a circuit such that its constituent nodes are either all honest or all corrupt. Thus any circuit uniquely decomposes into multiple segments (alternating honest and corrupt) and we can refer to, say, the first honest segment or the second corrupted segment in a circuit. Here the order of segments is understood to start from and include the proxy.

Challenge validity $(\mathcal{W}_0, \mathcal{W}_1, \mathcal{C})$. The predicate VALID$(\mathcal{W}_0, \mathcal{W}_1, \mathcal{C})$ checks that the adversary's choice of networks does not allow a trivial win. A fair number of conditions are checked for this purpose, where we additionally disallow some settings where, without loss of generality, an adversary could achieve the same advantage while adhering to our restrictions (if corruptions were adaptive, these simplifications would be less clean). We list the conditions and their justifications below.

1. **The two circuit vectors $\mathcal{W}_0$ and $\mathcal{W}_1$ contain the same number of circuits, i.e. $|\mathcal{W}_0| = |\mathcal{W}_1|$.**

The interface which we provide to the adversary for interacting with the network allows it to easily infer the number n of circuits present in the network; mainly through its oracles and by inspecting the states of the nodes that it controls. While for sufficiently large networks this may be hard to determine in practice, we do not aim to conceal this information through cryptographic means.

2. **Every circuit in $\mathcal{W}_0$ and $\mathcal{W}_1$ contains at most two corrupted segments.**

This restriction keeps the complexity of the security definition manageable. The consequence of this assumption is that the most complicated honest–corrupt configuration for a circuit will be two corrupted segments and up to three honest segments, with one of these honest segments sandwiched between the corrupt nodes. This middle honest segment will play an import role. For circuits consisting of a proxy and three routers—the default circuit length in Tor and sufficient for a minimal working example of the tagging attack [24]—the restriction is without loss of generality.

3. **For each i, circuits $\mathcal{W}_0[i]$ and $\mathcal{W}_1[i]$ share a subpath $[v_1, v_2, \ldots, v_{m-1}, v_m]$ where v_2 is the first corrupted node in either circuit, nodes $v_2 \ldots, v_{m-1}$ are corrupted, and either v_m is honest or it is the last node in both $\mathcal{W}_0[i]$ and $\mathcal{W}_1[i]$.**

When we introduce the Enc oracle, an adversary will be able to select a circuit with an honest proxy and ask for a message to be encrypted. The resulting ciphertext will be processed by the honest routers before a cell is handed to one of the routers under adversarial control. While the specific path a circuit index points to depends on which network the adversary is interacting with, this condition ensures that a message encrypted for circuit i will reach the adversary on the same edge and, where applicable, reenters the honest component identically, irrespective of the challenge bit b.

4. **For a given circuit vector consider the multiset of subpaths $[v_1, v_2, \ldots, v_{m-1}, v_m]$ where nodes v_1 and v_m are honest and nodes $v_2 \ldots, v_{m-1}$ are corrupted. Then the corresponding multisets for $\mathcal{W}_0$ and $\mathcal{W}_1$ should be identical.**

An adversary can always infer the directed subcircuits overlapping with the nodes it has corrupted, by observing to which state component (w) a cell gets associated to at each corrupted node. Thus the two networks are required to match on these directed subcircuits, including the adjacent honest nodes.

5. **For all i, if either $\mathcal{W}_0[i][0] \in \mathcal{C}$ or $\mathcal{W}_1[i][0] \in \mathcal{C}$ then $\mathcal{W}_0[i] = \mathcal{W}_1[i]$.**

If a circuit's proxy is corrupted in either of the two worlds, then the corresponding circuit must be the same in both networks. The rationale is that we assume that a proxy's state reveals the entire path of routers for each of the circuits it is involved in.

Altogether the conditions so far ensure that any information that is inevitably leaked through the corrupted nodes is identical in both worlds; the final two conditions are simplifying conditions.

6. **For all i there exists $j > 0$ such that $\mathcal{W}_0[i][j] \notin \mathcal{C}$.**

Every circuit must contain at least one honest router. If all routers in a circuit were corrupted (possibly with an honest proxy), by condition 3 the circuit (including the proxy) must be identical in both networks. The inclusion of such circuits does not benefit the adversary, as can be shown by a straightforward reduction, so for simplicity we assume that every circuit includes at least one honest router.

7. **Every circuit in $\mathcal{W}_0$ and $\mathcal{W}_1$ contains at least one router in $\mathcal{C}$.**

An adversary has very little control over a circuit consisting entirely of honest nodes: while it could trigger the encryption oracle, it wouldn't actually be able to observe any of the cells travelling on that circuit (as all the connections between honest nodes are protected). Moreover, for schemes that satisfy trajectory integrity, the creation and operation of honest circuits has no influence on

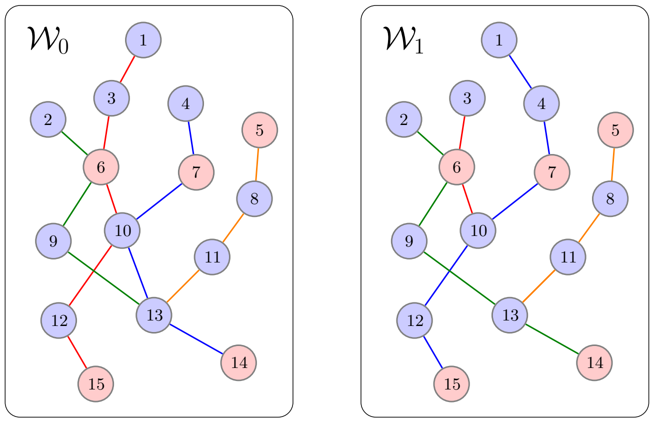

Fig. 6: An example of a valid challenge, where $\mathcal{W}_0 = [\,[2,6,9,13],[1,3,6,10,12,15],[4,7,10,13,14],[5,8,11,13]\,]$, $\mathcal{W}_1 = [\,[2,6,9,13,14],[3,6,10],[1,4,7,10,12,15],[5,8,11,13]\,]$, and $\mathcal{C} = \{5,6,7,14,15\}$ (marked in red).

the rest of the network (cryptographically speaking). Therefore, without loss of generality, we assume that the network does not contain all-honest circuits.

As each circuit has to contain at least one honest node and one corrupt node, for all circuits $\mathbf{p}$ in the C-HIDE game we will have that $|\mathbf{p}| \geq 2$.

An example of a valid challenge for the circuit-hiding game is depicted in Figure 6. Both circuit vectors contain 4 circuits, each of which contain at most two corrupted segments. In this particular case the proxy of circuit $[5,8,11,13]$ is corrupt and accordingly it is identical in both worlds. The encryption oracle can be queried on any of the first three circuits (i.e. $i \in \{1,2,3\}$), as their proxies are honest. In $\mathcal{W}_0$ the onion proxies corresponding to the three indices are 2, 1, and 4, whereas in $\mathcal{W}_1$ they are 2, 3, and 1. However in either case the corresponding cells are returned on the same edges, i.e. $(2,6),(3,6),(4,7)$. The Net oracle consists of the subgraph containing the nodes $8,9,10,11,12,13$ and can be accessed through four input edges $(6,9),(6,10),(7,10),(5,8)$ and two output edges $(12,15),(13,14)$. Note that while the internal structure of this subgraph differs between the two cases, the interface that the adversary sees, i.e. the set of input and output edges, is identical in both worlds, as required.

The Init-Circ procedure. For each circuit in $\mathcal{W}_b$ this procedure calls G to create initial states for the proxy and routers involved. Additionally some book-keeping is performed similar to prior games. Novel are the two sets of circuit indices, $\mathcal{I}_{\text{EN}}$ and $\mathcal{I}_{\text{NOP}}$, that Init-Circ keeps track of, for later use by the Net oracle.

The set $\mathcal{I}_{\text{EN}}$ contains the indices of all circuits which have an honest proxy and contain an *entry edge* (the predicate EN returns true), namely an edge from a corrupt node to an honest node.

```
Game C-HIDE$_{\mathsf{OE}}^{\mathcal{A}}$
  $(\mathcal{W}_0, \mathcal{W}_1, \mathcal{C}, \mathsf{st}) \leftarrow \mathcal{A}_1$
  if $\neg$ VALID$(\mathcal{W}_0, \mathcal{W}_1, \mathcal{C})$
    return false
  $\forall i\ \mathsf{sync}_i \leftarrow \mathsf{true}$
  $\varrho \leftarrow \varepsilon$; $n \leftarrow 0$; $b \leftarrow\!\!\$\ \{0,1\}$
  INIT-CIRC$(\mathcal{W}_b)$
  $\tau_{\mathcal{C}} \leftarrow \{(v, \boldsymbol{\sigma}_v, \boldsymbol{\tau}_v, \bar{\boldsymbol{\tau}}_v) \mid v \in \mathcal{C}\}$
  $b' \leftarrow \mathcal{A}_2^{\text{ENC},\text{NET}}(\mathsf{st}, \tau_{\mathcal{C}})$
  return $b = b'$

NET$(\mathbf{z})$
  $\forall i\ \mathsf{assc}_i \leftarrow 0$; $\mathbf{x} \leftarrow [\,]$
  for $i' = 1$ to $|\mathbf{z}|$
    $(s, v, c) \leftarrow \mathbf{z}[i']$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    if $s \notin \mathcal{C} \vee v \in \mathcal{C} \vee w = \perp$
      return $\lightning$
  for $i' = 1$ to $|\mathbf{z}|$
    $(s, v, c) \leftarrow \mathbf{z}[i']$; $c^* \leftarrow c$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    $(\bar{\boldsymbol{\tau}}_v[w], d, c) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
    $(i, j) \leftarrow \mathsf{map}(v, w)$
    while $d \notin \mathcal{C} \wedge d \neq \oslash$
      $s \leftarrow v$; $v \leftarrow d$
      $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
      $(\bar{\boldsymbol{\tau}}_v[w], d, c) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
    if $d \in \mathcal{C}$
      $\mathbf{x}.\mathsf{append}(v, d, c)$
    if $d \in \mathcal{C} \vee i \in \mathcal{I}_{\text{NOP}}$
      $\mathsf{assc}_i \leftarrow \mathsf{assc}_i + 1$
      if $c^* \neq Q^i.\mathsf{dequeue}()$
        $\mathsf{sync}_i \leftarrow \mathsf{false}$
  if $\bigvee_{i \in \mathcal{I}_{\text{EN}}} (\mathsf{sync}_i \vee \mathsf{assc}_i \neq 1)$
    return $\lightning$
  return $\mathsf{sort}(\mathbf{x})$

INIT-CIRC$(\mathcal{W})$
  for $i = 1$ to $|\mathcal{W}|$
    $n \leftarrow n + 1$; $\mathbf{p}_n \leftarrow \mathcal{W}[i]$
    $(\varrho, \sigma, \mathbf{t}, \bar{\mathbf{t}}) \leftarrow \mathsf{G}(\varrho, \mathbf{p}_n)$
    $\ell_n \leftarrow |\mathbf{p}_n|$
    $\mathsf{sync}_n \leftarrow \mathsf{true}$
    $\boldsymbol{\sigma}_{\mathbf{p}_n[0]}.\mathsf{append}(\sigma)$
    for $j = 1$ to $\ell_n$
      $v \leftarrow \mathbf{p}_n[j]$
      $\boldsymbol{\tau}_v.\mathsf{append}(\mathbf{t}[j])$
      $\bar{\boldsymbol{\tau}}_v.\mathsf{append}(\bar{\mathbf{t}}[j])$
    if EN$(\mathbf{p}_n, \mathcal{C}) \wedge \mathbf{p}_n[0] \notin \mathcal{C}$
      $\mathcal{I}_{\text{EN}} \leftarrow \mathcal{I}_{\text{EN}} \cup \{i\}$
      if NOP$(\mathbf{p}_n, \mathcal{C})$
        $\mathcal{I}_{\text{NOP}} \leftarrow \mathcal{I}_{\text{NOP}} \cup \{i\}$
  foreach $v$
    Shuffle$(\boldsymbol{\sigma}_v, \boldsymbol{\tau}_v, \bar{\boldsymbol{\tau}}_v)$

ENC$(i, m)$
  $(v, w) \leftarrow \mathsf{map}(i, 0)$
  if $v \in \mathcal{C}$
    return $\lightning$
  $(\boldsymbol{\sigma}_v[w], d, c) \leftarrow \mathsf{E}(\boldsymbol{\sigma}_v[w], m)$
  while $d \notin \mathcal{C}$
    $s \leftarrow v$; $v \leftarrow d$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    $(\bar{\boldsymbol{\tau}}_v[w], d, c) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
  $(v^*, d^*, c^*) \leftarrow (v, d, c)$
  while $d \in \mathcal{C}$
    $s \leftarrow v$; $v \leftarrow d$
    $w \leftarrow \mathsf{D}(\boldsymbol{\tau}_v, s, c)$
    $(\bar{\boldsymbol{\tau}}_v[w], d, c) \leftarrow \bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}_v[w], s, c)$
  if $d \neq \oslash$
    $(i, j) \leftarrow \mathsf{map}^{-1}(v, w)$
    $Q^i.\mathsf{enqueue}(c)$
  return $(v^*, d^*, c^*)$
```

Fig. 7: The C-HIDE game used to define circuit-hiding security for onion encryption.

The set $\mathcal{I}_{\text{NOP}}$ is a subset of $\mathcal{I}_{\text{EN}}$. It contains those circuits in $\mathcal{I}_{\text{EN}}$ for which the adversary is unable to observe an output. Specifically, the predicate NOP returns true iff after the first entry edge the circuit contains no corrupted nodes, i.e. the circuit contains only one corrupted segment. Thus an adversary may inject cells into these circuits through the entry edge, but lacking a later corrupted segment, it is unable to 'catch' the processed cells. Note however that such circuits are still highly relevant in the C-HIDE game as the adversary should not be able to infer which cells produce no output.

The Enc oracle. This oracle allows the adversary to encrypt a message m under any circuit i whose proxy is honest. As the adversary can only observe edges where one of the constituent nodes is corrupted, it will only get the ciphertext as output by the proxy if it happens to control the node which the proxy forwards it to. Otherwise we need to progress the ciphertext through the honest part of the circuit, until hitting a corrupted router. The first **while** loop takes care of this progression, resulting in a cell c^* and edge (v^*, d^*) that will be returned to the adversary.

However, before doing so, there is some further bookkeeping to be done on behest of the NET oracle. Recall that we allow up to two corrupt segments per circuit, so presumably after the corrupt segment starting with d^* the circuit can turn honest, and then corrupt again. In other words, there will be an honest middle segment. The NET oracle will allow the adversary to query these honest middle segments for all circuits simultaneously. Clearly, given ENC's interface, the adversary will know what messages are concealed across the various cells. Then it can always forward these to the NET oracle, where the cells it returns (at the interface between the honest segments and the second corrupt component) will correspond to a subset of the original set of messages. However, the adversary does not know which messages reside in which cells, though figuring this out would trivially identify the circuit over which the cell was sent (e.g. by embedding i in the message). To prevent trivial wins when an adversary also controls exit nodes (and is therefore able to recover plaintext) the NET oracle will suppress certain queries, based on the bookkeeping that ENC is about to do.

In the second **while** loop, the cell c^* is progressed further along the corrupted segment until the first honest node is encountered. Here the premise is that the nodes in the corrupted segment behave honestly (where the cells are processed using the routers' original state variables $\bar{\boldsymbol{\tau}}_v[w]$ as the adversary has its own separate states).

This process allows us to predetermine the cell c that the first honest node d will receive from the corrupted segment. Unless we already reached the end of the circuit (in which case $d = \oslash$), we add this cell to a queue Q^i corresponding to the circuit i. The queue will be used by the NET oracle to detect when cells sent by corrupt routers digress from these stored values. Once this happens, an adversary has become active with respect to that circuit.

The Net oracle. Finally, we give the adversary oracle access to the honest component of the network that it can inject traffic into. These are the honest circuit segments that are preceded by corrupted nodes. In turn these honest circuit segments may lead into a second corrupted segment or remain honest until the end. Hence, the NET oracle may return less cells than it receives in its input. Circuits with a corrupted proxy are also accessible through the NET oracle, but since they must be identical in both worlds, their corresponding output edges will be known to the adversary.

We will impose a number of restrictions on the NET oracle. After all, if the adversary could query each honest circuit segment individually, it could distinguishing the two networks simply by observing on which edge the corresponding cell is received. For this reason, the adversary can only query the honest circuit segments in parallel, so the sandwiched honest component behaves a little like a mix net.

Moreover, as already mentioned when discussing ENC, if the adversary is able to forward the cells obtained from ENC straight into NET (without modification), decryption of the resulting cells (using corrupted routers) would again allow to distinguish the two networks. Accordingly we restrict the oracle to only return an output when all honest circuit segments are queried in parallel and are all *out of sync*. The flag sync_i keeps track of whether circuit i is still *in sync* or whether it has gone *out of sync*. The exact meaning of a circuit being in or out of sync will be explained shortly.

Throughout, the adversary may query an honest circuit segment individually, but the output will be suppressed. This allows the adversary to progress the states of the routers along a particular circuit, prior to making the next 'parallel' query.

After intercepting cells through the ENC oracle, the adversary can manipulate, replay and re-order these cells and re-inject them into the honest part of the network through the NET oracle. As its input it takes a vector $\mathbf{z}$ of triples, each identifying a cell together with the edge on which it is incident. The first **for** loop verifies that this input satisfies two conditions. The first is that all cells be incident on an edge which the adversary has access to, that is, an entry edge. The other condition is that all cells must be associated to some circuit (i.e. $w \neq \perp$) by the honest node of the entry edge on which they are incident. If both checks are successful, the second **for** loop progresses each cell through the honest segments of their respective trajectories. In every iteration it stores the initial cell c^* and the circuit index i to which c^* was associated by the first processing node v. Every cell is progressed along its trajectory until it reaches a corrupted node or its destination. If the cell has reached a corrupted node, the cell and the corresponding output edge are stored in the output vector $\mathbf{x}$. In addition if the cell has reached a corrupted node or the circuit produces no output ($i \in \mathcal{I}_{\text{NOP}}$), the assc and sync variables for the associated circuit are updated. The variable assc_i keeps track of how many ciphertexts are associated to circuit i in a single oracle call. On the other hand, sync_i keeps track of whether the adversary has become active with respect to circuit i. This is determined by comparing c^* with

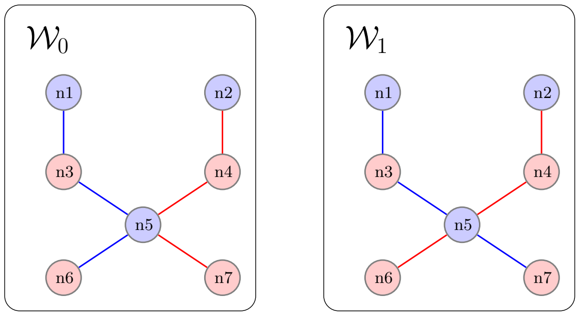

Fig. 8: The challenge used in the tagging attack example of Section 5.1, where $\mathcal{W}_0 = [\,[n1, n3, n5, n6], [n2, n4, n5, n7]\,]$, $\mathcal{W}_1 = [\,[n1, n3, n5, n7], [n2, n4, n5, n6]\,]$, and $\mathcal{C} = \{n3, n4, n6, n7\}$ (marked in red).

the next available ciphertext in the queue corresponding to circuit i. If these don't match, sync_i is set to false, indicating that the circuit went out of sync. Once a circuit goes out of sync, it stays out of sync.

It is important to note the conditions under which we update these variables. In particular, if a circuit contains two entry edges, a cell will only affect these variables if it has been injected through the first entry edge of that circuit. Clearly cells injected through the second entry edge will produce no output either.

An output is returned to the adversary only if *every* circuit in the set $\mathcal{I}_{\text{EN}}$ has exactly one cell in $\mathbf{z}$ associated to it and is out of sync. The first condition stops the adversary from correlating the endpoints merely through the number of cells that are input and output at each end. The latter condition is analogous to the suppression of output in stateful security definitions [1, 5, 10, 22]. On the other hand, circuits that have a corrupted proxy or whose routers are either all corrupted or all honest are excluded from this requirement (since we quantify over $\mathcal{I}_{\text{EN}}$).

Finally, before returning the output to the adversary we sort its components lexicographically to prevent the adversary from correlating the outputs with the inputs based on the ordering in which they have been processed by NET.

Definition 4. *The circuit hiding advantage of adversary $\mathcal{A}$ against* OE *is defined by*

$$\mathbf{Adv}^{\text{C-HIDE}}_{\mathsf{OE}}(\mathcal{A}) = 2 \cdot \Pr\left[\, \text{C-HIDE}^{\mathcal{A}}_{\mathsf{OE}} \Rightarrow \mathsf{true} \,\right] - 1 \ ,$$

where the game C-HIDE *is given in Figure 7.*

5.1 Capturing Tagging Attacks

One of our main goals was to arrive to an anonymity definition that captures tagging attacks, we now confirm that this is indeed the case. Consider Tor's current onion encryption scheme, based on counter-mode AES, described in Section 2.1. This scheme is not C-HIDE secure as evidenced by the following attack. The adversary outputs the challenge described in Figure 8. For any arbitrary message m it makes two encryption queries, $(1, m)$ and $(2, m)$, obtaining in return the respective replies, $(n1, n3, c_1)$ and $(n2, n4, c_2)$. The adversary progresses these cells past nodes $n3$ and $n4$ respectively, using its own copy of the nodes' states, to obtain the respective cells c_3 and c_4. It then tags both cells by flipping the last bit and last two bits of each cell respectively, that is, it queries $[(n3, n5, c_3 \oplus 1), (n4, n5, c_4 \oplus 11)]$ to the NET oracle. Since the cells' headers are unchanged each circuit will have exactly one cell associated to it, in addition both circuits will be out of sync because both cells have been modified by the adversary. Thus the oracle's output will not be suppressed and it will be of the form $[(n5, n6, c_5), (n5, n7, c_5')]$. At this point the adversary attempts to untag the cells and process them at their respective exit nodes, i.e. $c_5 \oplus 1$ at node $n6$ and $c_5' \oplus 11$ at node $n7$. If both decrypt correctly the adversary outputs 0 as its guess and 1 otherwise. It is easy to see that the adversary's advantage is very close to 1; the only time it fails is when both $c_5 \oplus 10$ and $c_5' \oplus 10$ decrypt correctly, which happens with low probability.

6 Preventing Tagging Attacks

On an intuitive level it is evident that tagging attacks in Tor are enabled by the inherent malleability of counter-mode encryption which carries on across multiple layers of encryption. Proposal 202 [31] identified two potential ways of addressing this. One approach would be to borrow from mix-net designs by appending a MAC tag to each encryption layer, where after tag verification each node would re-pad the cell to its original length [16,17]. The other was to replace counter mode encryption with a Variable-Input-Length (VIL) tweakable cipher, as used in disk encryption, which impedes malleability without incurring any ciphertext expansion. Clearly the increased space efficiency of this latter approach is a huge bonus, but back in 2012 all known VIL tweakable cipher constructions were significantly slower than counter mode encryption. This changed however with the advent of AEZ [26] whose efficiency is comparable to that of counter mode AES, albeit at the expense of a more heuristic security analysis. Now having a viable instantiation, the Tor project put forward a concrete design for a new onion encryption scheme in proposal 261 [32]. We will refer to the onion encryption scheme described therein as Tor261. We emphasize that a VIL tweakable cipher is only a building block, and constructing a secure onion encryption scheme from it is substantially non-trival. The rest of this section is devoted to put the security of Tor261 on firm grounds, but we first describe Tor261 in more detail.

6.1 VIL Tweakable Ciphers and AEZ

As the name suggests, a VIL tweakable cipher is a tweakable cipher that can operate over inputs of varying length. More precisely it is a pair of deterministic algorithms (Π, Π^{-1}) each of which takes a key K, a tweak tw and a string x, respectively y, to return a string y, respectively x, where $|x| = |y|$ and for all K and tw, $\Pi(K, tw, \cdot)$ is a permutation and $\Pi^{-1}(K, tw, \cdot)$ is its inverse. Recall that in Tor the cell size is fixed to 509 bytes and it would therefore suffice to have a tweakable cipher that can handle inputs of this length. Accordingly the term wide-block tweakable cipher is often used instead, but in reality all known constructions admit inputs of varying length. In terms of security, a tweakable cipher is expected to be a (strong) tweakable pseudorandom permutation. We refer the reader to [41] for an up-to-date introduction to VIL tweakable ciphers.

Technically, AEZ embodies a different primitive called Robust Authenticated Encryption (RAE) [26]. An RAE is a pair of deterministic algorithms (Π, Π^{-1}) where Π takes a key K, a nonce no, associated data ad, a message x and a stretch τ to return a ciphertext y of length $|x| + \tau$. The decryption algorithm Π^{-1} inverts this operation, taking a K, a nonce no, associated data ad and a ciphertext y to return either a message x, if y was generated honestly, or the special symbol $\perp$ indicating that y is invalid. When the key is chosen uniformly at random security requires that for any nonce and associated data, (Π, Π^{-1}) should behave as a pseudorandom injection, and its inverse, from binary strings to τ-bit longer ones. It is easy to see that if we set $\tau = 0$ RAE collapses to a VIL tweakable cipher, where the nonce and the associated data, collectively play the role of the tweak. Indeed in Tor proposal 261 τ is set to zero and we will therefore treat AEZ as a VIL tweakable cipher where the tweak is represented by the pair (no, ad).

6.2 Tor261: The Onion Encryption Scheme in Tor Proposal 261

The onion encryption scheme Tor261 is obtained by instantiating the Tor relay protocol [19] with the layer encryption described in [32]. Note that proposal 261 only affects the relay protocol and in particular the cryptography used in the circuit extend protocol is unaffected. A pseudocode description of Tor261 is displayed in Figure 9.

In addition to a VIL tweakable cipher, the scheme also makes use of a block cipher BC (instantiated with AES) in a Davies-Meyer-type configuration to compute a chain value h (by means of a separate chain key L) that is included in the tweak of every layer of encryption, i.e. Π evaluation. It is intended to provide forward security rather than anonymity or standard channel security, consequently it does not surface in our analysis. In addition, the tweak also contains the xor of the input and output strings from the *previous* layer encryption call. Intuitively this serves to create a domino effect whereby the corruption of any cell will corrupt all subsequent cells. The no component of the tweak is composed of a counter ctr and two binary flags, fwd and $early$, encoded as single-byte strings. These indicate respectively the direction of travel of the cell with respect to the direction in which the circuit was established, and whether the cell is of the type RELAY or RELAY_EARLY (the two cell types handled by the relay protocol). Whether a cell is of type RELAY or RELAY_EARLY is indicated in the command field (cmd) in the cell header, through byte values 3 and 9 respectively.

```
algorithm $\mathsf{G}(\varrho, \mathbf{p})$
  $\ell \leftarrow |\mathbf{p}|, ctr \leftarrow 0, \mathbf{c}_0 \leftarrow [\varepsilon]_1^{\ell}$
  $(v^*, d^*) \leftarrow (\mathbf{p}[0], \mathbf{p}[1])$
  do $id_o \leftarrow_{\$} \{0,1\}^{32}$ until $(v^*, d^*, id_o) \notin \varrho$
  $\varrho \leftarrow \varrho \cup \{(v^*, d^*, id_o)\}$
  $id_o^* \leftarrow id_o$
  for $j = 1$ to $\ell$
    $(s, v, d) \leftarrow (\mathbf{p}[j-1], \mathbf{p}[j], \mathbf{p}[j+1])$
    $\mathbf{K}[j] \leftarrow_{\$} \{0,1\}^k$
    $\mathbf{L}[j] \leftarrow_{\$} \{0,1\}^{256}$
    $\mathbf{h}[j] \leftarrow_{\$} \{0,1\}^{128}$
    $id_i \leftarrow id_o$
    do $id_o \leftarrow_{\$} \{0,1\}^{32}$ until $(v, d, id_o) \notin \varrho$
    $\varrho \leftarrow \varrho \cup \{(v, d, id_o)\}$
    $\mathbf{t}[j] \leftarrow (s, id_i)$
    $\bar{\mathbf{t}}[j] \leftarrow (v, \mathbf{K}[j], \mathbf{L}[j], \mathbf{h}[j], ctr, \varepsilon, \varepsilon, d, id_o)$
  $\sigma \leftarrow (\ell, \mathbf{K}, \mathbf{L}, \mathbf{h}, ctr, \mathbf{c}_0, d^*, id_o^*)$
  return $(\varrho, \sigma, \mathbf{t}, \bar{\mathbf{t}})$

algorithm $\mathsf{E}(\sigma, m)$
  $early \leftarrow 0$
  parse $\sigma$ as $(\ell, \mathbf{K}, \mathbf{L}, \mathbf{h}, ctr, \mathbf{c}_0, d, id_o)$
  $\mathbf{c}_1[\ell] \leftarrow \mathsf{encode}(m)$
  for $j = \ell$ to $1$
    $no \leftarrow \langle ctr \rangle_{64} \parallel \langle fwd \rangle_8 \parallel \langle early \rangle_8$
    $\mathbf{h}[n] \leftarrow \mathsf{BC}(\mathbf{L}[j], \mathbf{h}[j]) \oplus \mathbf{h}[j]$
    $ad \leftarrow (\mathbf{c}_0[j] \oplus \mathbf{c}_0[j-1]) \parallel \mathbf{h}[j]$
    $\mathbf{c}_1[j-1] \leftarrow \Pi(\mathbf{K}[j], (no, ad), \mathbf{c}_1[j])$
  if $early = 1$ : $cmd \leftarrow \langle 9 \rangle_8$
  else : $cmd \leftarrow \langle 3 \rangle_8$
  $\hat{c} \leftarrow (id_o, cmd, \mathbf{c}_1[0])$
  $ctr \leftarrow ctr + 1$
  $\sigma \leftarrow (\ell, \mathbf{K}, \mathbf{L}, \mathbf{h}, ctr, \mathbf{c}_1, d, id_o)$
  return $(\sigma, d, \hat{c})$

algorithm $\mathsf{D}(\boldsymbol{\tau}, s, \hat{c})$
  parse $\hat{c}$ as $(id_i, cmd, c)$
  if $cmd \neq \langle 3 \rangle_8$
    return $\perp$
  for $w = 1$ to $|\boldsymbol{\tau}|$
    if $(s, id_i) = \boldsymbol{\tau}[w]$
      return $w$
  return $\perp$

algorithm $\bar{\mathsf{D}}(\bar{\boldsymbol{\tau}}[w], s, \hat{c})$
  parse $\bar{\boldsymbol{\tau}}[w]$
    as $(v, K, L, h, ctr, c_0, c_0', d, id_o)$
  parse $\hat{c}$ as $(id_i, cmd, c_1)$
  if $cmd = \langle 9 \rangle_8$ : $early \leftarrow 1$
  else : $early \leftarrow 0$
  $no \leftarrow \langle ctr \rangle_{64} \parallel \langle fwd \rangle_8 \parallel \langle early \rangle_8$
  $h \leftarrow \mathsf{BC}(L, h) \oplus h$
  $ad \leftarrow (c_0 \oplus c_0') \parallel h$
  $c_1' \leftarrow \Pi^{-1}(K, (no, ad), c_1)$
  $ctr \leftarrow ctr + 1$
  if $\mathsf{chkzeros}(c_1') \wedge d \neq \perp$
    $x \leftarrow \mathsf{decode}(c_1')$; $d^* \leftarrow \emptyset$
  else
    if $d \in \{\perp, \emptyset\}$
      $d \leftarrow \perp$; $d^* \leftarrow \emptyset$; $x \leftarrow \perp$
    else
      $d^* \leftarrow d$
      $x \leftarrow (id_o, cmd, c_1')$
  $\bar{\boldsymbol{\tau}}[w] \leftarrow (K, L, h, ctr, c_1, c_1', d, id_o)$
  return $(\bar{\boldsymbol{\tau}}[w], d^*, x)$
```

Fig. 9: The Onion Encryption Scheme Tor261 and its variant $\overline{\text{Tor261}}$. Scheme $\overline{\text{Tor261}}$ includes the shaded code but Tor261 does not.

Thus during decryption the *early* flag is set according to the value described in the cell's command field. In all other cases we treat fwd and $early$ as internal variables set in accordance with the context in which encryption and decryption are operating. Since we only consider unidirectional anonymous channels in the forward direction, $fwd = 1$, always.

In our modelling of the Tor relay protocol we make the following assumptions and simplifications. We assume a version 4 or higher cell format with a total cell size of 514 bytes. In addition to the cell header relay cells include a payload header and proposal 261 alters the format of the payload header. However the only cryptographic processing of this header is limited to checking that the redundancy in certain fields is correctly formatted. Specifically it identifies 55 bits that should be verified to contain zeros upon decryption, but suggests that this verification could be extended to other fields for added security. We model the processing of a message by padding it and prepending the relay header through an encode function. Similarly during the decryption we will employ a decode function to reverse this process and a function $\mathsf{chkzeros}$ to verify that the relevant fields contain zeros. For generality, in our analysis we assume the number of bits set to zero by encode and later verified by $\mathsf{chkzeros}$ is r.

As before, a node determines to which circuit it should associate a cell from the circuit identifier id in the cell header as described in Section 2.1. Circuit identifiers are chosen during circuit establishment by the various nodes involved, which we abstracted in the circuit creation algorithm G. Except for a mechanism to avoid collisions, the Tor specification does not specify how circuit identifiers are to be chosen. Since it is not particularly relevant for our analysis, for simplicity we assume these are sampled by G uniformly at random without replacement with respect to every edge. In particular, G maintains a state ϱ comprised of a set of triples (a, b, id) to keep track that the circuit identifier id is in use on edge (a, b) and thereby avoid collisions.

The Tor specification allows for certain messages to be delivered to nodes in the circuit other than the last one. As far as we are aware, this functionality is not actually used in the relay protocol in practice. Our syntax does not allow an onion proxy to specify cells for an intermediate node, but intermediate nodes may nonetheless recognise a cell as being intended for them in $\mathsf{Tor261}$. Thus from the perspective of an intermediate node a cell is either recognised or else it is forwarded along the circuit, but it is never deemed invalid. On the other hand, if the last node in a circuit receives a cell which it does not recognise, it declares the cell as invalid and the circuit is torn down [20]. To model this behaviour we overload the semantics of the variable d in the node's decryption state component. For any circuit, it is intended to store the next hop in the circuit with respect to the current node and is set to $\oslash$ if the node is last in the circuit. When the last node detects an invalid cell it sets $d \leftarrow \perp$ to indicate such an event, and returns an error for that cell and any subsequent ones. However the other nodes in the circuit are unaffected since the adversary may be able to block the cells instructing the circuit teardown.

6.3 Circuit Hiding

As described in Section 6.2 the relay protocol supports two types of cells, RELAY and RELAY_EARLY. The sole purpose of RELAY_EARLY cells is to enable a mechanism for limiting the length of circuits in Tor, see [19]. While the details of this mechanism are beyond our scope, partly because it extends over to the circuit extend protocol, the support of RELAY_EARLY cells in Tor261 exposes it to tagging attacks. In essence an adversary can tag a cell by flipping the *cmd* field in its header which will then propagate along the circuit unaltered. As an example consider the challenge depicted in Figure 8. After making two Enc queries, one on each circuit, the adversary forges a RELAY cell on $(n3, n5)$ and a RELAY_EARLY cell on $(n4, n5)$, and submits both as a single Net query. If the cell output on $(n5, n7)$ is RELAY_EARLY it knows it is interacting with $\mathcal{W}_1$.

Tagging attacks manipulating the *cmd* field were already exploited in the infamous 2014 incident [2], and as such this vulnerability is a real concern and not just an artifact of our security definition. Interestingly, Tor261 appears to attempt to protect against this by including the *early* flag in the nonce *no* but as we just pointed out this does not prevent the attack. It could be argued however that this attack is somewhat limited in practice since it only admits one type of tag. On the other hand, the current onion encryption scheme allows an adversary to tag each cell with a unique mark allowing it to de-anonymise multiple circuits in parallel. This limitation could potentially be overcome by instead tagging a unique mark in a *sequence* of cells. However this possibility is limited by the fact that in Tor honest nodes are required to tear down the circuit if they observe more than eight RELAY_EARLY cells. Moreover, in a typical setting three RELAY_EARLY cells would already have been used up during circuit establishment. Thus in practice an adversary would have at most 2^5 unique tags at its disposal. On the one hand, this improves significantly over Tor's current state of affairs but it still falls short of the best possible security.

Unable to prove Tor261 secure we consider its variant $\overline{\mathsf{Tor261}}$, also described in Figure 9, which supports only RELAY cells and prove that it meets our circuit hiding notion. This serves to show that the above attack is the only way of mounting tagging attacks on Tor261, which could possibly be prevented by adopting an alternative mechanism, not involving RELAY_EARLY cells, for limiting circuit size or mitigated further by reducing the maximum circuit size. Informally, Theorem 1 states that $\overline{\mathsf{Tor261}}$ is circuit hiding as long as (Π, Π^{-1}) is a secure VIL tweakable cipher in the $\pm\widetilde{\mathrm{prp}}$ sense [26, 41], and m and r are sufficiently large.

Theorem 1 ($\overline{\mathsf{Tor261}}$ is Circuit Hiding). *Let $\overline{\mathsf{Tor261}}$ be the scheme described in Figure 9 composed of a VIL tweakable cipher (Π, Π^{-1}) and an encoding scheme* encode *that prepends messages with r zeros. Then for any circuit hiding adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ running in time t, making q_e queries to* Enc *and q_n queries to* Net*, there exists a $\pm\widetilde{\mathrm{prp}}$ adversary $\mathcal{B}$ running in time t' and making q_f and q_i queries to its forward and inverse oracles, such that:*

$$\mathbf{Adv}^{\text{C-HIDE}}_{\overline{\mathsf{Tor261}}}(\mathcal{A}) \leq |\mathcal{I}_{\text{EN}}| \left(2\,\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\Pi,\Pi^{-1}}(\mathcal{B}) + 2^{-r+1} + \min(q_e, q_n)\, 2^{-m+1} \right),$$

where $\mathcal{I}_{\text{EN}}$ is the set of circuits with an honest proxy that the adversary can inject cells into (i.e. circuits containing an entry edge). Furthermore we have that $q_f \leq q_e$, $q_i \leq q_n$, and $t' \leq t + q_e \ell_{max} T + q_n |\mathcal{I}_{\text{EN}}|(\ell_{max} - 1)T$ where ℓ_{max} is the maximum length of a circuit and T is the maximum time needed to evaluate Π or Π^{-1}.

The full proof of Theorem 1 can be found in the full version [18]. Below we outline the main intuition.

Proof outline. We prove the theorem through a standard game-hopping technique. We start with the C-HIDE game instantiated with $\overline{\mathsf{Tor261}}$ and focus on the honest node in the entry edge (if any) of each circuit. We then replace the VIL tweakable cipher instance corresponding to that node with a truly random permutation. We gradually chop parts of the circuits until we eventually end up with a game that depends solely the state information pertaining to those subpaths that are required (by VALID) to be common to both worlds (corresponding to the two possible values of b). It can then be shown that for $\overline{\mathsf{Tor261}}$ the state information corresponding to these subpaths is identically distributed in either world. Thus we eventually end up with a game that is independent of the bit b in which case the adversary can do no better than guess the bit b.

7 Conclusion

Motivated by Tor's susceptibility to tagging attacks and the ongoing effort in the Tor community to thwart them, we initiated a formal treatment of circuit-based onion encryption. In our treatment, we opted for a level of abstraction that is closer to practice than previous works. For instance, we explicitly included the routing functionality that characterizes onion routing. While this choice arguably complicates security definitions and analysis, we believe it provides for a more informative model, allowing us to expose certain hitherto unsuspected conflicts between routing and security. One illustration is the potential for correlating traffic between onion router based on the seniority of circuit, which has implications for the data structure used to store a router's full decryption state.

We analysed Tor's new proposal Tor261, intended to prevent tagging attacks, using our new framework. Our analysis confirms that its overall design is sound, yet also exposes that its support of RELAY_EARLY cells still enables tagging attacks. Presently, we focused on the circuit-hiding property of the proposed scheme, leaving open the analyis of Tor261's end-to-end channel security.

Finally, unidirectional channels are only a first step in the formal analysis of Tor and related onion routing protocols. Marson and Poettering [30] recently exposed important challenges when composing ordinary unidirectional secure channels and it is unclear how those challenges affect anonymous channels. (This cautionary remark does not refer to any specific issue that we identified, rather it delineates our analysis.) Furthermore, extending our analysis to include dynamic circuit establishment and deal with adaptive corruptions, are challenging open problems.

Acknowledgments. We would like to thank Matthew Green for suggesting this problem to us and Jonathan Katz for helpful initial discussions. We are indebted to Nick Matthewson for clarifying certain historical and practical aspects of Tor. We also thank the anonymous reviewers for their constructive feedback.

Degabriele was supported in part by EPSRC grant EP/M013472/1 (UK Quantum Technology Hub for Quantum Communications Technologies) and in part by the German Federal Ministry of Education and Research (BMBF) within CRISP.

References

1. Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1480–1491. ACM Press, October 2016.
2. Roger Dingledine (arma). Tor security advisory: "relay early" traffic confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack, July 2014.
3. Michael Backes, Ian Goldberg, Aniket Kate, and Esfandiar Mohammadi. Provably secure and practical onion routing. In *CSF*, pages 369–385. IEEE Computer Society, 2012.
4. Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009.
5. Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 02*, pages 1–11. ACM Press, November 2002.
6. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000.
7. Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 312–329. Springer, Heidelberg, August 2012.
8. Daniel J. Bernstein, Mridul Nandi, and Palash Sarkar. HHFHFH. Dagstuhl, 2016.
9. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. Cryptology ePrint Archive, Report 2016/1188, 2016. http://eprint.iacr.org/2016/1188.
10. Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 682–699. Springer, Heidelberg, April 2012.
11. Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 169–187. Springer, Heidelberg, August 2005.

12. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, August 2003.
13. Sambuddho Chakravarty, Marco Valerio Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *PAM 2014*, volume 8362 of *LNCS*, pages 247–257. Springer, 2014.
14. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
15. George Danezis, Claudia Diaz, and Paul Syverson. Systems for anonymous communication. *CRC Handbook of Financial Cryptography and Security*, page 61, 2009.
16. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE Computer Society Press, May 2003.
17. George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 IEEE Symposium on Security and Privacy*, pages 269–282. IEEE Computer Society Press, May 2009.
18. Jean Paul Degabriele and Martijn Stam. Untagging Tor: A formal treatment of onion encryption. Cryptology ePrint Archive, Report 2018/XXX, 2018. https://eprint.iacr.org/2018/XXX.
19. Roger Dingledine and Nick Mathewson. Tor protocol specification. https://gitweb.torproject.org/torspec.git/plain/tor-spec.txt, —.
20. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.
21. Joan Feigenbaum, Aaron Johnson, and Paul F. Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Trans. Inf. Syst. Secur.*, 15(3):14:1–14:28, 2012.
22. Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 545–564. Springer, Heidelberg, August 2015.
23. Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In Vijayalakshmi Atluri, editor, *ACM CCS 02*, pages 193–206. ACM Press, November 2002.
24. Xinwen Fu and Zhen Ling. One cell is enough to break Tor's anonymity. In *Proceedings of Black Hat DC 2009*, 2009. 10 pages.
25. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Information Hiding*, volume 1174 of *LNCS*, pages 137–150. Springer, 1996.
26. Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015.
27. The invisible internet project (I2P). https://geti2p.net.
28. Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. Users get routed: traffic correlation on tor by realistic adversaries. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 337–348. ACM Press, November 2013.

29. Brian Neil Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems (extended abstract). In Ari Juels, editor, *FC 2004*, volume 3110 of *LNCS*, pages 251–265. Springer, Heidelberg, February 2004.
30. Giorgia Azzurra Marson and Bertram Poettering. Security notions for bidirectional channels. *IACR Trans. Symm. Cryptol.*, 2017(1):405–426, 2017.
31. Nick Mathewson. Proposal 202: Two improved relay encryption protocols for Tor cells. https://lists.torproject.org/pipermail/tor-dev/2012-June/003649.html, June 2012.
32. Nick Mathewson. Proposal 261: AEZ for relay cryptography. https://lists.torproject.org/pipermail/tor-dev/2015-December/010080.html, December 2015.
33. Steven J. Murdoch and Piotr Zielinski. Sampled traffic analysis by internet-exchange-level adversaries. In *Privacy Enhancing Technologies (PET 2007)*, volume 4776 of *LNCS*, pages 167–183. Springer, 2007.
34. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002.
35. The23rd Raccoon. How I learned to stop ph34ring NSA and love the base rate fallacy. http://archives.seul.org/or/dev/Sep-2008/msg00016.html, September 2008.
36. The23rd Raccoon. Analysis of the relative severity of tagging attacks. http://archives.seul.org/or/dev/Mar-2012/msg00019.html, March 2012.
37. Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for anonymous routing. In *ACSAC'96*, pages 95–104. IEEE Computer Society, 1996.
38. Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with MorphMix. In Ari Juels, editor, *FC 2004*, volume 3110 of *LNCS*, pages 233–250. Springer, Heidelberg, February 2004.
39. Phillip Rogaway and Yusi Zhang. Onion-AE: Foundations of nested encryption. Cryptology ePrint Archive, Report 2018/126, 2018. https://eprint.iacr.org/2018/126.
40. Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS 2003*, volume 2808 of *LNCS*, pages 116–131. Springer, Heidelberg, October 2003.
41. Thomas Shrimpton and R. Seth Terashima. A modular framework for building variable-input-length tweakable ciphers. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 405–423. Springer, Heidelberg, December 2013.
42. Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy*, pages 44–54. IEEE Computer Society Press, 1997.
43. Paul F. Syverson, Gene Tsudik, Michael G. Reed, and Carl E. Landwehr. Towards an analysis of onion routing security. In *Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 96–114. Springer, 2000.