Generic, Decentralized, Unstoppable Anonymity:

The Phantom Protocol

1. Abstract

Recent years, and especially this past year, have seen a notable upswing in developments toward anti-online privacy around the world, primarily in the form of draconian surveillance and censorship laws (both legislated and suggested) and ISPs being pressured into individually acting as both police and informants for various commercial interests. Once such first steps have been taken, it is of course also of huge concern how these newly created possibilities could be used outside of their originally stated bounds, and what the future of such developments may hold in store for online privacy.

There are no signs of this trend being broken anytime soon. Combined with the ever growing online migration of everything in general, and privacy sensitive activities in particular (like e.g. voting, all nature of personal and interpersonal discussions, and various personal groupings), this trend will in turn unavoidably lead to a huge demand for online anonymization tools and similar means of maintaining privacy.

However, if not carefully designed, such anonymization tools will, ultimately, be easy targets for additional draconian legislation and directed [il]legal pressure from big commercial and political interests. Therefore, a well-conceived, robust and theoretically secure design for such an anonymization protocol and infrastructure is needed, which is exactly what is set out to be done with this project.

What is presented in this paper is the design of a protocol and complete system for anonymization, intended as a candidate for a free, open, community owned, de facto anonymization standard, aimed at improving on existing solutions such as e.g. TOR – from the viewpoint of the needs of today and tomorrow – and having the following important main properties and goals:

1. Completely decentralized.
   - No critical or weak points to attack or put [il]legal pressure on.
2. Maximum resistance against all kinds of DoS attacks.
   - Direct technical destructive attacks will be the only possible practical way to even attempt to stop it.
3. Theoretically secure anonymization.
   - Probabilistic methods (contrary to deterministic methods) must be used in a completely decentralized design like this, where no other peer can be trusted, so focus is put on optimizing these methods.
4. Theoretically secure end-to-end transport encryption.
   - This is simple in itself, but still important in the context of anonymization.
5. Completely (virtually) isolated from the "normal" Internet.
   - No one should have to worry about crimes being perpetrated from their own IP address.
6. Maximum protection against identification of protocol usage through traffic analysis.
   - You never know what the next draconian law might be.
7. Capable of handling larger data volumes, with acceptable throughput.
   - Most existing anonymization solutions are practically unusable for (or even prohibit) larger data volumes.
8. Generic and well-abstracted design, compatible with all new *and existing* network enabled software.
   - Software application developer participation should not be needed, it should be easy to apply the anonymization to both new and already existing products like e.g. web browsers and file transfer software.

The Phantom protocol has been designed to meet all these requirements, and will be presented in this paper.

TABLE OF CONTENTS

TABLE OF FIGURES

2. Introduction — Anonymity in the Context of This Paper

Anonymity can mean a lot of different things depending on the context in which it is used. In order to avoid any misunderstandings in this paper, we will begin with a definition of anonymity in the context at hand, which at the same time just might give some insight into other aspects and motivations of this paper too.

Anonymity at its simplest core could be described as the inability for (all) other parties to discover, in a given context, the *identity* of the party defined as being anonymous.

Well then you say, what is the definition of *identity*? In the context of this paper, and most other contexts dealing with individual human beings, the identity of a person could be defined as the set of information that directly, conclusively and uniquely describes and singles out that individual among all other human beings in this world (and possibly other worlds too, depending on one's religious preferences). That is, more practically, the information needed to be able to show up on a particular person's doorstep, or at least file a (non-John Doe) lawsuit against them.

The subject of "John Doe lawsuits", in turn (which is a law suit directed at a party whose real identity has not yet been discovered) brings us to the subject of *pseudonymity*, which is related to, and could perhaps be called "a lower form" of, anonymity. Under pseudonymity, a party can operate without revealing its real identity, but various acts performed can still all be connected and bound to this same entity, i.e. to its pseudonym identity. This pseudonym identity, in turn, runs the risk of being connected to the real identity at some later point in time, and, if and when that occurs, any act already connected to the pseudonym identity will be instantly attached to the real identity, even though this real identity was not known at the time the acts were committed.

Going one step further in defining the concept of anonymity in the context of this paper, which in turn deals primarily with anonymity in the context of Internet communications, we present here below examples of some different practical and theoretical levels of anonymity that exist on the Internet today:

- For those most blissfully ignorant Internet users, the Internet seemingly offers complete anonymity. As far as these are concerned, you can just register a Hotmail address under an imaginary name, an ICQ account with a naughty handle, an Internet dating site account with a picture of David Hasselhoff, and then start enjoying your new and alternative identity, without anyone ever being able to expose you or do the first thing to prevent it.
  — Complete anonymity. Diplomatic immunity of the Internet kind, woohoo!
- At the next level of enlightenment, the first problems of *pseudonymity* start to become apparent to the previously so blissfully ignorant Internet user. It actually turns out that if you start pumping out Viagra spam from your Hotmail or ICQ account, or start to sexually harass just a few too many people over at that Internet dating site, your über anonymous email, ICQ and Internet dating accounts will be reported to their individual service providers, who will shut them down and maybe even send an angry letter to that other ultra anonymous email address that you were just about to start using for your latest body part enlargement product marketing campaign. Your *pseudonyms* have been tracked down and taken to justice for your actions, and you now think you're starting to catch on to what that local EFF activist was going on and on about before you slammed the door in his face.
  — Replaceable pseudonymity. Ok, whatever, you can always register another email account.
- At yet the next level of enlightenment, just when your latest massive Hotmail-borne Cialis spam campaigns and your email trojan hobby project of finding naked pictures of people on their home computers were starting to take off, it seems like some evil ultra haxor have tracked down something called your "IP address", and sent a bunch of abuse reports to all those nice service providers that make your fledgling business enterprise so enjoyable. Now all of a sudden you can't seem to register any more batches of Hotmail

addresses with that cool Russian program, nor register any more Internet dating accounts with pictures of David Hasselhoff. And to top it off, that nosy ISP of yours even had the nerve to send a rude warning letter to your home, saying something about unacceptable behavior, service agreements and other legal mumbo jumbo. Where the hell do they get off saying something like that, not to mention how on earth could they know anything about what you've been doing in the first place? What, do they have spies looking in on people's computer screens though their windows now? After pulling down all the curtains, moving the radio-button in the cool Russian program from "Hotmail" to "Gmail", and finally managing to find another Internet dating site that apparently didn't find it affordable to pay the membership fee to the international black list of dating site weirdos, pervs and just plain too ugly people, you really start to regret that you didn't at least give the EFF guy the opportunity to move his face out of the way before slamming the door, realizing now that he might actually have been on to something there.
— Non-replaceable pseudonymity, the ISP seems to have some magical way of knowing what you do, but luckily they aren't allowed to reveal your real identity to anyone else, right? (the ISP has logs of all your allotted IP addresses, which can be matched to abuse reports)

- It is said that the Internet is a place of constant learning, and the field of anonymity is no exception to that rule. What started out as a simple and righteous retaliation campaign against the rude people behind those service providers trying to shut down your legitimate businesses in the fields of Cialis retail and amateur porn, quickly turned into a passionate hobby, and even yet another successful Internet enterprise. This after it became apparent that the combination of email death threats and extortion by means of private nude pictures is surprisingly lucrative. All of a sudden though, while minding your own business as usual, planning your new life on Hawaii, it once again becomes abruptly apparent that you have been the victim of one of those horrible online first amendment crimes that you have a faint recollection of being the last thing to ever pass by the original front teeth of that EFF guy. All of a sudden, your rights within the area of speaking have apparently been radically cut down to the somewhat disappointing "right to remain silent", and you start wondering what the world has really come to? Well, actually, you knew it would happen eventually, just like it happened to all great Internet enterprises, the "unfair competition" lawsuit trick, the oldest trick in the book.
— Partial lack of anonymity/pseudonymity. People seem to somehow be able to get hold of your real identity by suing your Internet identity, or otherwise reporting it to the police.

- Just returned from the week long visit "downtown", you are starting to seriously consider a small donation to the EFF, since installing that hard disk encryption software mentioned in the blood stained pamphlet left by the EFF guy outside your door seemingly resulted in a failure for your competitors to plant fake evidence of unfair competition on your computer, or at least that's your interpretation of the legal mumbo jumbo that ended with "dismissed due to lack of evidence". So it really worked then it seems, those EFF people really managed to save your freedom of speech after all, and they deserve a donation for it indeed! Being a responsible donor though, you always make sure to check up on the financial activities and transactions of the organizations to which you donate money. After all, you don't want to have any part in sponsoring something that would later turn out to be some kind of shady operation, right? Incidentally, this would also be a great opportunity to try out that latest service offering of your most fierce competitor in the area of Internet entrepreneurship, namely the Russian Business Network. A service which they have given the disquietingly inventive name "Get the bank account transaction listing of any given company or private person, for 10 cents". Availing yourself of this service and having skimmed through the EFF account transaction lists, you don't notice anything out of the ordinary except one small oddity. Two local chapters of Alcoholics Anonymous (which you conclude, from their initials, must be the Rock Island and Menlo Park AA chapters) have repeatedly made some pretty huge donations to key personnel at EFF, all of which have been immediately "reimbursed" shortly thereafter on each occasion. Poor people you think, they must have

> been drunk and mistyped the account numbers when transferring money from the local chapters to the national organization. Naturally, being the Good Samaritan that you are, you decide to take a quick peek at the individual account listings of these “RIAA” and “MPAA” local chapters, just to insure they didn’t misplace any more of their money while being under the influence lately. You quickly get to regret this thoroughness though, when seeing that these poor fellows seem to have misplaced money transfers to practically every single ISP board member, judge or politician involved in questions concerning copyright and digital communications law in the whole of the United States and Europe. That’s just too much, you say to yourself, it would take all night to note down and report all those mistaken transactions. Those poor drunkards are on their own for now. Damn, alcohol abuse is crazy, almost as crazy as all those policies and laws proposed by the exact ISPs, politicians and judges being present on this transaction listing, regarding data retention, blocking of traffic to arbitrary destinations, and release of personal traffic logs to third-party make-believe cops owned by commercial interests. He-he, isn’t that a funny coincidence you think to yourself, before finally registering your $10 donation to the EFF, which after all seems to be a bunch of some pretty standup guys, having returned all those misplaced donations and everything.
> *— Full lack of anonymity, and subsequent invasion of privacy and abuse of personal information by commercial interests and other shady authorities could soon pose a huge threat to Internet users all around the world (as a result of misplaced AA money transfers if nothing else).*

So, can’t anything be done about this current and impending state of affairs?

Yes it can! In addition to donating money to charitable organizations (e.g. the EFF) that work actively with helping “AA chapters” all over the world to stay away from their abuse, an efficient way of removing the possibility for such privacy invasion and abuse would be to completely remove the compromising link between a network information exchange and the network identification details of the peers involved in such exchanges, i.e. the IP addresses of the communicating peers.

This, combined with removing the possibility of eavesdropping on any part of the contents of such information exchanges, would insure that no external party whatsoever could neither see who is talking to whom, nor discover what they are talking about, be it the ISP itself or any other organization tapping the wire at any given point along the communication path.

Oh, and about the danger of “over entrepreneurial” individuals (like our dear friend from the examples above) exploiting this anonymity to commit serious crimes like those in the examples above, don’t worry, this has been taken into consideration and taken care of in the design of the protocol too, in one of the most clean and beautiful of ways possible.

And *that*, dear readers, is the definition of anonymity in the context of this paper.

3. Further Definitions and Limitations

3.1. Design Assumptions

Yet again, for a topic as multifaceted as this, some important assumptions on which the design is based should be declared:

1. The traffic of every node in the network is assumed to be eavesdropped on (individually, but *not* globally in a fully correlated fashion) by an external party.
2. Arbitrary random peers participating in the anonymization network/protocol are assumed to be compromised and/or adverse.
3. The protocol design cannot allow for any trusted and/or central entity, since such an entity would always run the risk of being potentially forced offline or manipulated, if for no other reason due to large enough quantities of money (or lawyers) being “misplaced”.

3.2. Important Consequences of Design Goals and Assumptions

As a result of the design assumptions listed above, a number of consequences critical to the design of the protocol can be deduced, out of which some important ones are:

1. The protocol needs to be fully decentralized and distributed.
2. No other peer participating in the protocol can be trusted by itself.
3. Probabilistically secure algorithms need to be used instead of deterministically secure ones.

3.3. Design Directives

During the course of any design procedure, one is often faced with different alternatives or options at different levels. Design directives are meant to assist in making these decisions, and also to make sure that all such decisions are made in a consistent manner, toward the same higher level goals. The most important design directive for this project is:

1. CPU power, network bandwidth, working memory and secondary storage resources are all relatively cheap, and will all be available in ever increasing quantities during the coming years and thereafter. Thus, wherever a choice must be made between better security or better performance/lower resource consumption, the most secure alternative should be chosen (within reasonable bounds, of course).

4. Design Goals of the Protocol

4.1. Complete Decentralization

In the world of IT security, it is a well known fact that however efficient, secure and robust a system or protocol may be, if it has any kind of single weak point, this is what any determined attacker will find and exploit. In cases of involvement by big financial and/or political interests, an additional common angle of attack is the legal arena, where lawsuits, legal threats and economical pressure are the primary weapons of choice.

Thus, an important design goal of the protocol is to make sure that it has no central or single weak points, neither in the IT realm, nor the legal realm. Because of this, the only viable option is to make the protocol completely decentralized (including legal ownership wise, i.e. also making it open source and community owned), dependent only on the sum of its users, while at the same time not specifically on any single instance of them.

4.2. Maximum Resistance against All Kinds of DoS Attacks

The current widespread trend of anti-privacy measures around the world is there for a reason. Big commercial and political interests, and combinations of them, want to restrict and control the privacy and free exchange of information on the Internet today. Considering the decentralized nature of this protocol, combined with the fact that its primary purpose is actually to prevent the tracking and targeting of single individuals on the basis of who they are communicating with and why, the only practical way to attempt to stop it would be to launch destructive technical attacks of different kinds against the network as a whole (attempting to outlaw its use altogether would of course also be a theoretical possibility, which is discussed separately later in this paper).

Thus, an important design goal of the protocol is to be resistant against such attacks, in the best way possible, and this goal should therefore be considered through all levels of its design.

4.3. Theoretically Secure Anonymization

As always in the field of security, any solution relying solely on the obscurity and *practical* difficulty of cracking it, will always fail sooner or later. Usually sooner too actually, if just enough motivation and resources exist among its adversaries.

Thus, an important design goal of the protocol is that the security of its anonymity should be theoretically provable, regardless of being deterministic or probabilistic.

4.4. Theoretically Secure End-to-End Encryption

End-to-end encryption, and the subsequent prevention of anyone eavesdropping on the contents of single communication sessions, is something that is normally taken for granted on the Internet today. This kind of secrecy is also of extra importance when it comes to the field of anonymization, due to the simple fact that if someone is able to eavesdrop on the contents of the communication between two otherwise anonymous parties, it is highly likely that information of more or less identifying nature will occur at some point. In such case, the identities of the communicating parties can be deduced by means of this information, instead of through network specific address information.

Thus, an important design goal of the protocol is to make sure that no entity other than the two communicating peers of the protected conversation has access to its contents. Or, put in a simpler way, the enforcement of end-to-end encryption.

4.5. Complete (Virtual) Isolation from the "Normal" Internet

It is a well-known fact that people behave worse while under the cover of anonymity. Many opponents of a protocol of the type being presented here would argue that the Internet is sufficiently full of crime as it is today, and that increased anonymity would only be for the worse (apparently not considering anti-privacy crime).

A big disadvantage with many existing anonymity solutions, e.g. TOR, is that they only provide an anonymizing layer between their users and the normal, non-anonymous, Internet. This results in a design containing “out-proxies” for the last communication step between the anonymization layer and the normal Internet, which in turn has two substantial drawbacks. First of all, the traffic of this last communication step is not necessarily encrypted, which enables any out-proxy to eavesdrop on any such information (and as mentioned above, eavesdropping of communication contents is not only a confidentiality risk, it is also a direct anonymity risk!). Second of all, the users of such a system are essentially able to perform any action that normal non-anonymous Internet users can, including most kinds of Internet related crime, such as spamming and unlawful hacking. These types of crimes can always be traced back to the last “out-proxy” before the switch to normal non-anonymous Internet traffic, and the person acting as this out-proxy (which in a decentralized design would be any random user of the system) would run the risk of being associated with the crime in question. Both these factors are of course at the very least unpleasant, and would have a very discouraging effect on the usage of the system.

Thus, an important design goal of the protocol is to make sure that none of its users should need to worry about crimes being committed against other random Internet users from their own computer, which is in turn accomplished by complete isolation between the anonymized network and the normal Internet. This way, each and every user can decide exactly which services they want to expose to anonymous access, considering the risks associated with doing so. For example, if you don’t want to take the risk of anonymous hackers vandalizing your web server, simply don’t expose it to the anonymous network. If you don’t want to take the risk of anonymous users threatening you in a chat room, don’t join a chat room on the anonymous network. This way (possibly even in combination with legal measures like EULAs and licenses for the protocol and its implementations), no user of the anonymous network (let’s call this user “A”) should be able to blame another innocent user of the anonymous network (let’s call this user “B”) for an anonymous attack which just happened to bounce off user B as a last random stop before hitting user A. Thus, no user of the network should have to worry about their participating role as a possible routing node when using the network.

NOTE: Out-proxies from the Phantom protocol toward the normal non-anonymous Internet can still be implemented on the application level, and hosted by anyone willing to take the risk. In this case they could also easily be made target-specific, in order to e.g. allow for anonymous access to your own website only, or similar.

4.6. Maximum Protection against Protocol Identification/Profiling

Already today, several ISPs around the world have been exposed throttling or blocking traffic that they either don’t themselves like, or even worse, that some commercial or political third-party interests disapprove of and, thus, pressured them to block, all for a large variety of reasons. Due to the fact that many of these parties in many cases are the very ones whose actions in the field of anti-privacy this anonymization protocol aims to counteract, it is fair to assume that the protocol itself would quickly become an attractive target for such measures.

Thus, an important design goal of the protocol is to make positive identification of its use as hard as possible for any third party with full external eavesdropping access to its traffic, in order to prevent these same parties (e.g. an ISP) from being able to act upon this kind of information. Let’s not kid ourselves however, traffic analysis is extremely hard to defend against to the extent of it not being able to conclude *anything*. The practical goal will rather be to induce a large enough amount of uncertainty and false positives into any reasonably resourceful traffic analysis method, in order to prevent real-time throttling and blocking.

4.7. High Traffic Volume and Throughput Capacity

Practically all free and generally available anonymization solutions of today operate at speeds too low for most users to be interested, or at least be attracted to them. Many of these solutions also outright explicitly and completely forbid the transfer of larger data volumes. First of all, poor performance and limitations of any kind are negative traits of any protocol or system, by themselves. Second of all, and even more importantly, the strength of the provided anonymity (and also the robustness against DoS attacks of different kinds) in a fully distributed and decentralized solution like the Phantom protocol, depends on its number of users.

Thus, an important design goal of the protocol is to make it as capable, high-performing and non-limited as possible in the aspects of transfer volume and throughput, while still maintaining theoretically strong anonymization.

4.8. Generic, Well-Abstracted and Backward Compatible Design

Last but not least, there are some design goals that practically always constitute winning strategies when designing complex systems and powerful solutions. First of all, in the long run, a generic system is practically always superior to a specific system, and especially if it is popular and used by many people, thus becoming the subject of the accompanying constant flow of ideas and suggestions for new and creative ways of using and adapting it. Second of all, a well-abstracted design makes sure that a design mistake in one area doesn't affect unnecessarily large parts of the solution as a whole, and also allows for much more efficient collaborative work by different groups of people with different areas of expertise. Third of all, a design that is backward compatible with previous solutions and applications will get a much quicker start, has much greater opportunities to quickly show off its potential, and is much easier to put into a perspective that people understand and appreciate.

Thus, an important design goal of the protocol is to make it as generic, well-abstracted and backward compatible with existing adjacent and relevant technologies as possible. This might sound a bit vague, highfalutin and unnecessary to mention, but as you will see, it will be of great importance in many ways for this protocol.

5. Bird's-Eye View of the Protocol

Now, with all the major goals, directives and assumptions behind the protocol design having been presented, we can begin to delve into its actual design. At its core, and viewed at the highest level, the design of the protocol is very simple.

5.1. Some Initial Definitions

- A *protocol user* is any actor or device that is using the protocol. It could be a home user, a large server, or possibly even a dedicated anonymization appliance. Everything that has access to the Internet is a valid candidate.
- In the context of this protocol, the *real identity* of a protocol user is equated with its IP address on the network in question (which for all matters of this paper will be assumed to be the Internet). This will hold true even if this IP address isn't necessarily a dedicated one for the user, i.e. even if the user is one of many behind a particular NAT firewall, the identity of the user is considered to be revealed to a certain party if the external IP address of this NAT firewall is revealed to the same party.
- All protocol users participate in the *anonymous network*, or rather, the protocol users are even the sole constituents of the anonymous network, since it is fully distributed and decentralized.
- All protocol users in the anonymous network will be represented as *network nodes* in this paper. In explanatory figures, these nodes will in turn be represented as circles (of different color, depending on their role in the current explanation).
- An *anonymized node* is a network node that is making use of the anonymous network to hide its own identity while communicating with another node in the anonymous network (which in turn can be anonymized, or not, by its own choice).

5.2. A First Glance

Now, let's say we have a situation where a network node, *α*, wishes to be anonymous while communicating with another network node, *β*. The resulting situation when using the protocol can be illustrated with the following figure.

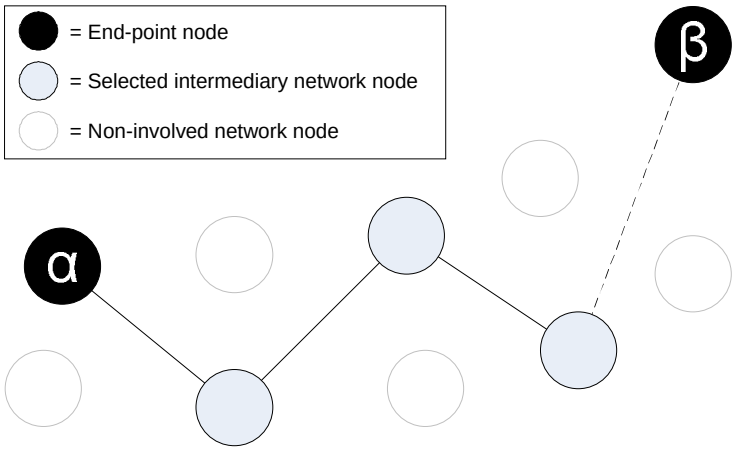

Figure 1. One-way anonymized communication, overview

As can be seen in the figure, the traffic from *α* is being forwarded through a number of intermediary nodes before reaching *β*, and any reply from *β* will be sent back through the same path.

This way, *β* has no way of knowing the identity of *α*, since *β* only knows the IP address of the last intermediary node, but has no knowledge of what is going on beyond that point.

5.3. A Little Further Look

As a necessary consequence of the protocol design (which is in turn based on the previously listed goals, directives and assumptions, and will be described in more detail later in this paper), all nodes in the network are responsible for maintaining their own anonymizing forwarding paths like this, or *routing paths* as they are called within the bounds of this protocol.

In the previous example, the routing path belonged to node *α*, and thus could only be used to anonymize node *α*. This means that, in this example, only node *α* was being anonymized. This is a perfectly valid circumstance though, where a node, in this case node *β*, has only joined the network to be able to *communicate with* anonymized nodes, while not itself being anonymized.

There are of course plenty of situations where *both* parties of a communication session want to remain anonymous, and the resulting situation can be illustrated with the following figure.

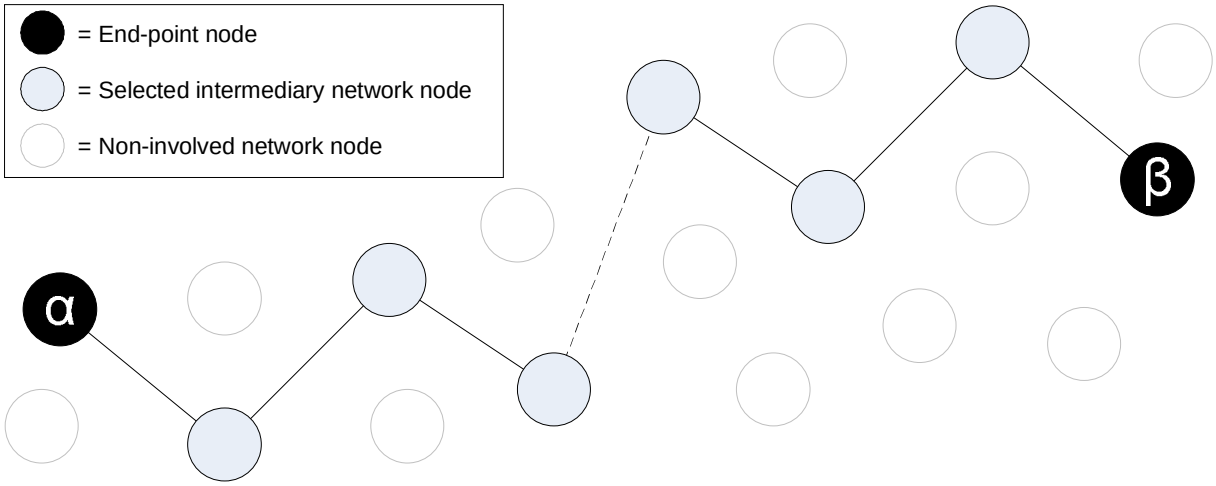

Figure 2. Two-way anonymized communication, overview

As can be seen in this figure, the traffic from node *α* is now being forwarded though a group of intermediary nodes, owned and controlled by *α*, after which it is sent to another group of intermediary nodes, owned and controlled by *β*, and then finally reaches node *β*. Any reply from *β* will be sent in reverse, back through the same path.

This way, *β* has no way of knowing the identity of *α*, since *β* only knows (at most) the IP address of the last intermediary node in the routing path owned by *α*, but has no knowledge of what is going on beyond that point. This time though, the exact same thing also applies in the reverse direction, i.e. *α* has no way of knowing the identity of *β*, since *α* only knows (at most) the IP address of the last intermediary node in the routing path owned by *β*, but has no knowledge of what is going on beyond that point.

We now have a simple model for a communication session where neither party has the ability to know the identity of the other party, while still being able to engage in unrestricted communication. This is practically what the Phantom protocol is about at its core, but as they say, the devil is in the details, and there are quite a bunch of those to consider.

6. High-Level Design

The previous chapter raises several questions, and also allows for a few more definitions. This chapter will try to clear up a few of these, while still on a higher, non-detailed, level.

6.1. Some Further Definitions

- A *routing path* is a number of network nodes in the anonymous network, in a defined order, selected by a particular anonymized node that also "owns" the path, over which communications to/from this anonymized node can be forwarded/routed in order to help keep its real identity hidden from its communication peers in the anonymous network.
- An *exit routing path*, or *exit route*, is a routing path through which the owning anonymized node can make outgoing connections to other nodes in the anonymous network, and thus, a mechanism for anonymizing network clients.
- The outermost node in an exit route is called an *exit node*.
- An *entry routing path*, or *entry route*, is a routing path through which the owning anonymized node can accept incoming connections from other nodes in the anonymous network, and thus, a mechanism for anonymizing network servers.
- The outermost node in an entry route is called an *entry node*.
- A *routing tunnel* is a connection established over a routing path, over which the anonymized node owning the routing path can perform TCP-equivalent anonymized communication with a specific peer node in the anonymous network.
- The *network database* is a fully distributed, decentralized database, based on DHT (distributed hash table[1]) technology. It contains a number of individual virtual "tables", which in turn contain all global information necessary for the operation of the anonymous network. All network nodes have access to the necessary parts of the contents of this database, through a well-defined API.
- An *AP address* (*Anonymous Protocol address*) is the equivalent of an IP address within the bounds of the anonymous network. AP addresses are used to identify individual nodes on the anonymous network (without for that sake being able to deduce their IP address or any similarly identifying real-world information).

6.2. Routing Paths

The concept of routing paths is the central anonymizing mechanism of the protocol. All network nodes wanting to hide their identity, i.e. the *anonymized nodes* of the network, make use of such routing paths, and the individual anonymized nodes are also fully responsible for setting up and maintaining these paths by themselves.

Routing paths can either be used to anonymize outgoing connections (e.g. clients connecting to a web server) or to anonymize incoming connections (e.g. a web server wanting to remain anonymous, while still allowing any client to connect to it), or both. This decision is also up to each individual network node itself to make. As mentioned in the definitions above, a routing path used to anonymize outgoing connections is called an *exit path*, and a routing path used to anonymize incoming connections is called an *entry path*.

The general idea behind a routing path is that any node in the network, let's call it α, is able to route its communications with any other node in the network through an arbitrary number of intermediary network nodes. These intermediary network nodes are selected by the anonymized node itself, which means that they are at worst probabilistically insecure (if random nodes are selected from the network, there is a risk that they are malicious and collaborating with other

[1] http://en.wikipedia.org/wiki/Distributed_hash_table

nodes out-of-band, in order to reveal the identity of other nodes), and at best fully trusted (i.e. if network nodes belonging to trusted friends are used).

The probabilistic insecurity factor is handled by using an arbitrary number of intermediary nodes, selected by the anonymized node itself. The protocol is also designed such that no intermediary node can derive the position at which it is located in the routing path, also not whether it is adjacent to the anonymized node itself or not, and finally not even in which direction of the path that the anonymized node is located. Also, the protocol is designed in such a way that no intermediary node can ever eavesdrop on any of the information being routed through it, by means of encryption.

Let's illustrate such a routing path with a figure.

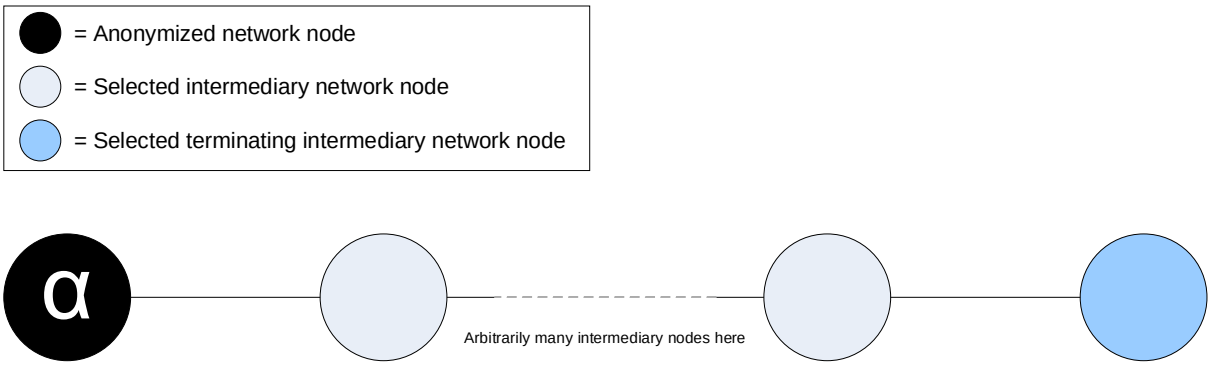

Figure 3. Routing path, overview

As can be seen in the figure, the anonymized node itself resides in one end of the routing path, and is then followed by an arbitrary number of intermediary nodes, before the routing path finally terminates in a special terminating intermediary node. If the routing path is an exit path, this terminating intermediary node is called an exit node, and if the routing path is an entry path, the terminating intermediary node is called an entry node. Both the number of nodes and the exact nodes themselves are selected by the anonymized node, when it constructs the routing path.

The terminating intermediary node has some special responsibilities in addition to the pure data routing task to which all intermediary nodes are dedicated. In the case of exit paths, the terminating node (i.e. the exit node) establishes all outbound connections to target AP addresses requested by the anonymized node that owns the routing path (thus being located at the other end of it). In the case of entry paths, the terminating node (i.e. the entry node) listens for incoming connections from other nodes in the anonymous network, and forwards these into the routing path in order to establish a full connection from the remote peer to the anonymized node that owns the routing path (thus being located at the other end of it).

As soon as such an outgoing or incoming connection is established, it becomes a separate routing tunnel inside the routing path. The same routing path can theoretically be used for an arbitrary number of parallel routing tunnels.

The real security of the protocol is of course intricately connected to the details of exactly how these routing paths are created by the anonymized nodes, and how they are found by other peers in the anonymous network. This will be discussed in detail in the following chapter, about low-level design. Before we get to that, however, there are still some components of the protocol left to discuss at this higher level.

6.3. Routing Tunnels

As mentioned in the definitions section above, a routing tunnel is the dedicated communication channel created inside a routing path as soon as a connection is successfully established between the anonymized node owning the routing path and another node on the anonymous network. Such a connection between two nodes on the network can be seen as the anonymized equivalent of a TCP connection over the common Internet.

It should be noted however, that the routing tunnel does not constitute the *entire* anonymized connection between two nodes on the network. Rather, it makes up only the half of the connection that belongs to the node owning the routing path over which the connection is being established. The connection then continues from the terminating intermediary node of the routing path, off to the other half of the connection, belonging to the other peer of the connection. This can be illustrated with the following figure.

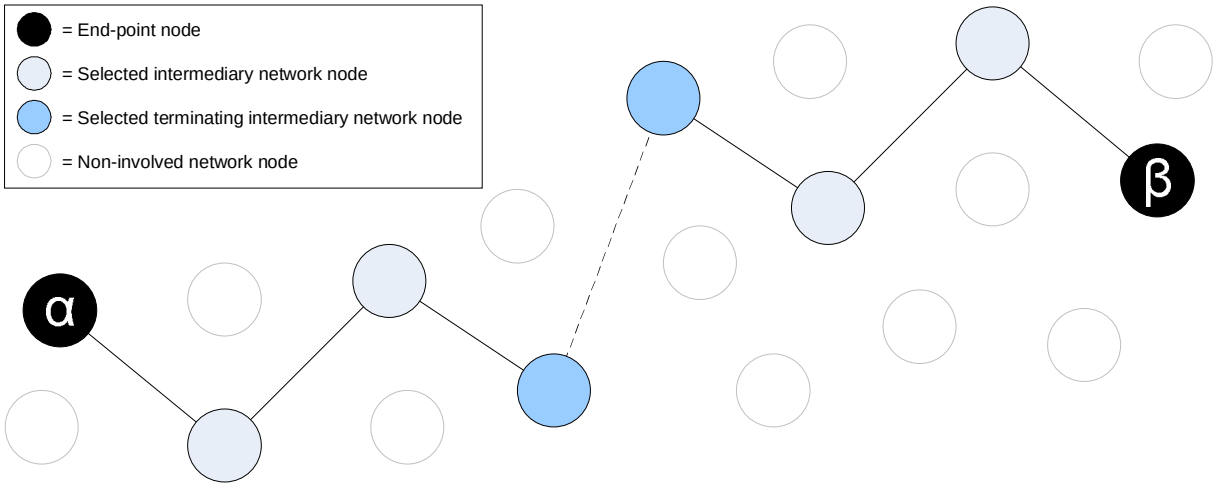

Figure 4. Two-way anonymized connection, involving two routing paths/tunnels

As can be seen in the figure, both of the end-point nodes of the connection are anonymized, and thus each has its own routing path. These two routing paths are then connected with each other by their respective exit/entry nodes, forming a single two-way anonymized connection, passing through two routing tunnels and a connecting link between them.

As described in the previous chapter, not all nodes in the anonymous network are necessarily anonymized though. Some of the nodes could just as well only be in the network to be able to *communicate with* other anonymized nodes, without for that matter having the need to anonymize themselves, thus saving resources and improving performance. Imagine for example a human rights discussion forum on a web server located in a free democratic country. Such a web server doesn't have any immediate reason to anonymize *itself*, while at the same time some of its users located in countries with repressive regimes might have an urgent need to both anonymize themselves with respect to the web server (out of fear that someone connected to the regime might get access to its log files) and also hide from their own ISP the fact that they are communicating with this web server at all. In this situation, the web server can join the anonymous network just in order to be able to accommodate users having these needs, but can still itself be completely without anonymization. This can be illustrated with the following figure, where the anonymized web client is represented by the anonymized node *α*, and the non-anonymized web server, still located on the anonymous network, is represented by the node *β*:

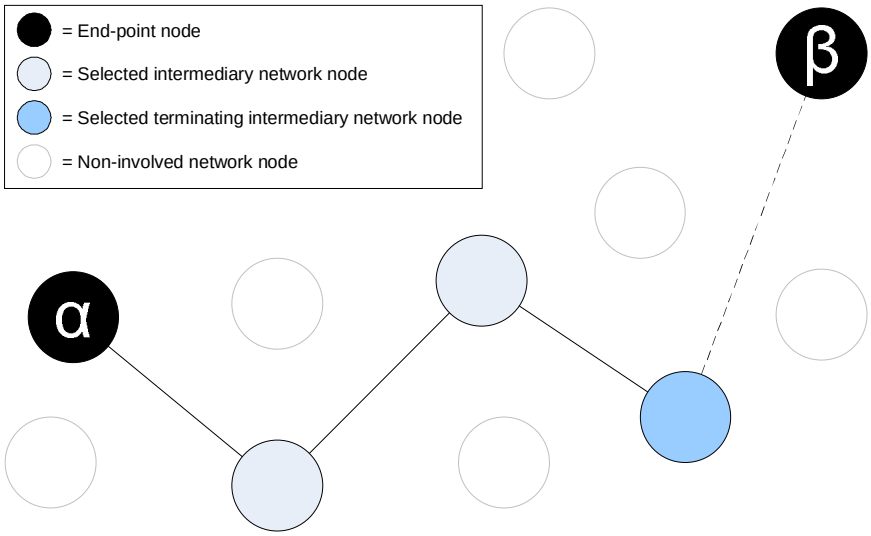

Figure 5. One-way anonymized connection, involving one routing path/tunnel

As can be seen in the figure, only the anonymized web client end-point node (*α*) of the connection has its own routing path. The exit node of this routing path then connects directly to the non-anonymized web server end-point node (*β*). Thus, the entire one-way anonymized connection constitutes a single routing tunnel, going over the single routing path, finally being connected directly to the non-anonymized end-point node (*β*) with a connecting link from the exit node of the exit path belonging to the anonymized node (*α*).

It should be noted however, that even though the end-point node (*β*) is not anonymized, it is still part of the anonymous network, reachable (only) through an AP address just like any other node in the network. Not only that, but the protocol is also designed in such a way that it should be impossible for any node (in this case *α*) to know or conclude if it is connected directly to a node (in this case *β*), or to a routing path owned by the same node. Thus, even nodes that don't make use of routing paths themselves can still in many ways be considered as being anonymized, if by no other means, through the existence of "reasonable doubt", which has very important performance implications for the protocol. This kind of reasonable doubt is, of course, of little use for a server with a static AP address, since it won't take long for any client to notice that the IP address of its "entry node", i.e. the server node itself in the non-anonymized case, is always the same. For various clients in different situations, however, this can be of great usefulness, discussed in more detail a bit later.

Anyway, the real security of the protocol is of course intricately connected to the details of exactly how these routing tunnels are established over the routing paths, connected with each other, and how data is routed through them. This will be discussed in detail in the following chapter, about low-level design. Before we get to that, however, there are still some components of the protocol left to discuss at this higher level.

6.4. AP Addresses

6.4.1. A Comparison between AP Addresses and IP Addresses

As mentioned in the definition section above, an AP address (Anonymous Protocol address) is the equivalent of an IP address within the bounds of the anonymous network. Just like with IP addresses on the Internet, each network node on the anonymous network has a unique AP address[2]. There is, however, one key difference between an IP address on the Internet today, and an AP address on the anonymous network. This difference lies in the connection between the network address used by a person or company, and their real identity.

With a common IP address on the Internet today, it is always theoretically possible to track down the real individual behind the address, or at least the legally responsible individual behind the address.

With an AP address in the anonymous network, no one should be able to infer any further information about the real identity of the node behind an address just by knowing the address itself. To put it another way, just because you can communicate with someone over the network, you shouldn't automatically be able to get hold of their real identity. This key difference between IP addresses and AP addresses is actually the entire reason for using the anonymization protocol in the first place, rather than just using normal Internet communication directly over standard TCP/IP.

Other than this key difference, AP addresses are actually extremely similar to IP addresses, for the reasons further discussed below.

6.4.2. Backward Compatibility with TCP/IP Enabled Applications

The structure of an AP address is identical to the structure of an IP address, i.e. it constitutes 32 bits, arranged in four period-separated 8 bit numbers, like "1.2.3.4". This equivalence to IP addresses is very important in order to maintain backward compatibility with all existing IP network communication enabled software, and even the possibility to anonymization-enable these applications without their own knowledge, without their source code, and without the participation of their authors.

This kind of third-party anonymization enabling of existing applications can be easily accomplished by the simple application of some binary network API hooks, as long as the application level interface for communicating over the anonymous network is identical to the application level interface for communicating over standard TCP/IP.

For this reason, AP addresses have been designed to permit just that, an interface identical to standard TCP/IP communications, and thus, AP addresses are structured just like IP addresses. In order for a node on the anonymous network to connect to another node on the anonymous network, a port number is also needed in addition to stating an AP address, just like with TCP connections, for the same backward compatibility reasons.

Do note however, that as IPv6 addresses become commonplace on the Internet, there's nothing in the design of the protocol that makes it hard to start supporting these too, in exactly the same way.

[2] Actually, only nodes wanting to accept *incoming* connections really need to have an AP address, even.

6.5. The Network Database

In order for a group of individual and completely autonomous network nodes to be able to accomplish anything useful together, or even to just be able to contact each other in the first place, some kind of common data store or communication channel would seem to be required. The problem with our restriction dictating a completely decentralized solution, however, is of course that it disallows central points of any kind in the network, e.g. a central directory server.

This has been solved by using a distributed and decentralized database, of the DHT (Distributed Hash Table) type. This way, the collected set of all nodes in the anonymous network actually *are* the database. Several successful large scale implementations of this kind of database already exist today, among which the Kademlia based Kad network database (of eMule fame) is one of the largest. Not only do most DHT database designs solve the basic problem of an efficiently distributed database in a good way, they also have built in resilience to the circumstance of constantly disappearing and newly joining nodes, and in some cases even resilience to malicious nodes attempting to inject corrupt data into the database.

All that is needed from a node in order to connect to, and become part of, the distributed database, and thus the anonymous network, is to get hold of any single node that is already connected. Without a central point, this might, at first glance, appear to be a big problem, especially for first-time users of such a network. It really isn't though. For any user that has already been connected to the network, even once, thousands upon thousands of node addresses can be cached until the next time the node wants to connect. At that point, significant numbers of these will most likely still be available, and, as already mentioned, contact with just *one single* connected node is all that is needed to become a fully qualified part of the database and network, thus again getting access to large volumes of node IP addresses to use as entry points for subsequent connections. Also, to create easily accessible entry points for never-before connected nodes, any node that has been connected to the network just once can easily publish and share its list of valid node IP addresses to the world in any number of ways, from their website, from their blog, in forums, blog comments, news group postings, or even by email. This guarantees that, as long as there are any remaining members in the anonymous network, an entry point is only a few mouse clicks or a URL away. Also, the DHT based database will be designed in such a way that separate, isolated segments, or "islands" of it should never be able to occur, or at least not exist for any longer periods of time.

This global network database will primarily contain two things:

- A table containing the IP addresses/ports of all currently connected nodes in the anonymous network (remember, it is not a secret that you are *connected to* the anonymous network, only *who* you are communicating *with* on this network, and *what* you are communicating!), and a set of accompanying properties for each such address.
- A table containing the AP addresses of all currently connected and/or registered nodes in the network, and a set of accompanying properties for each such address. It is very important to note that this table is *completely* decoupled from the table of IP addresses, which would of course otherwise completely defeat the purpose of the anonymization protocol to begin with. This table will among other things be used by network nodes to be able to find the entry nodes to be used when contacting other AP addresses.

The real security and stability of the protocol is of course intricately connected to the exact details of what data this database contains and how the database works and is implemented. This will be discussed in more detail in the following chapter, about low-level design.

7. Low-Level Design

Armed with the knowledge gained from the previous chapters, we now plunge yet another level deeper into the details of the protocol, taking a look at some of the very nitty-gritty details that make it tick.

7.1. Routing Paths

As mentioned above, the routing path is one of the key elements providing the anonymity in the protocol. As can be suspected, the exact procedure for setting up such routing paths in a secure way is of utmost importance to keeping the system theoretically secure. In this section, the path setup procedure will be described in more detail. We will begin with a shorter step-by-step description, and will then move on to describing and explaining each of the steps, and their underlying purposes and reasons, more thoroughly.

7.1.1. Secure Establishment of Routing Paths – Low-Level Overview

Without further ado, here follows the procedure for securely setting up a routing path:

1. The anonymized node (to be) selects a random set (with some exceptions to be discussed later) of x nodes from the IP address/port table of the network database, and fetches all their stored info, like e.g. their *path building certificate* (containing the public part of an asymmetrical encryption key-pair) and their *communication certificate* (containing a valid SSL certificate). These selected nodes will be called *X-nodes* from this point on, and are the ones that will constitute the final routing path.

Figure 6. Routing path creation, step 1

2. The anonymized node selects a similarly random set of y nodes from the IP address/port table of the network database, and fetches all their stored info, like e.g. their *path building certificate*, containing the public part of an asymmetrical encryption key-pair and their *communication certificate* (containing a valid SSL certificate). These selected nodes will be called *Y-nodes* from this point on.

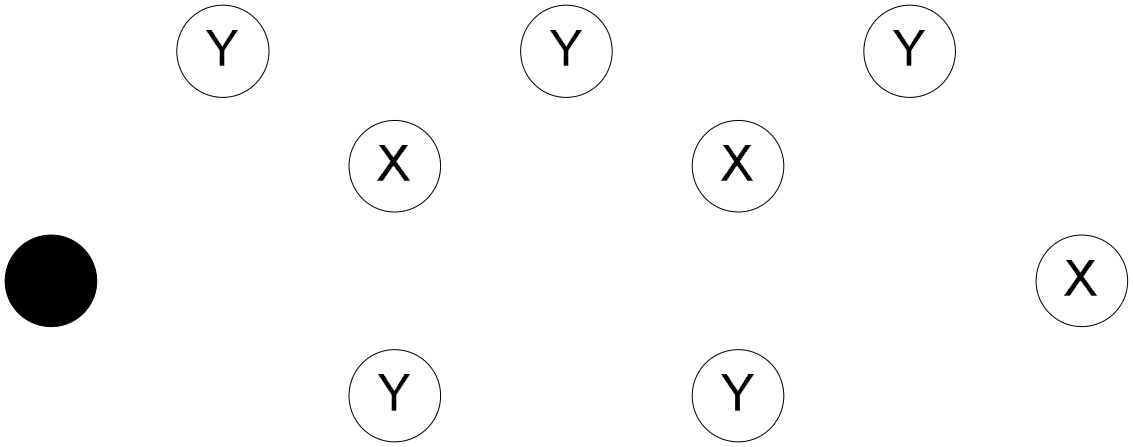

Figure 7. Routing path creation, step 2

3. The anonymized node chooses, arbitrarily, an ordered sequence made up of all the X-nodes and Y-nodes, being constructed in a way that:
   - No two X-nodes are adjacent to each other.
   - A Y-node is located in one end of the sequence.
   - A number of Y-nodes equal to the total number of X-nodes minus one (although always at least one), are located adjacent to each other in the other end of the sequence.
   - One end of the sequence is chosen at random to be the beginning of the sequence.

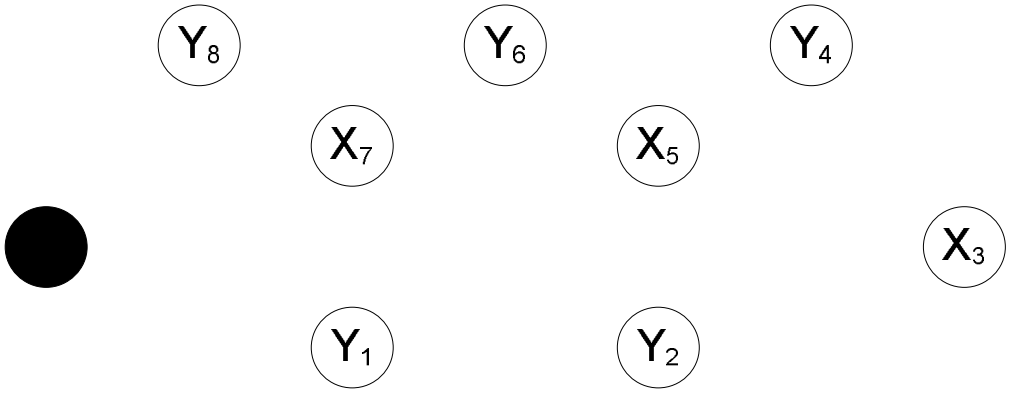

Figure 8. Routing path creation, step 3

4. The anonymized node generates a temporary asymmetrical cryptographic key-pair, to be used only during the setup procedure of this specific routing path. The private key of this key-pair will be called *the routing path construction key*, and the public key of this key-pair will be called *the routing path construction certificate*, from now on.
5. The anonymized node prepares a special unique *"setup package"* for each individual X-node and Y-node, being asymmetrically encrypted with the public key of the individual recipient, symmetrically encrypted with the 128-bit ID of its incoming connection, and signed with the routing path construction key from the previous step. The package contains the following (just as with all other steps, this will be explained in more detail in the next section):
   5.1. IP address of the expected previous node in the sequence.
   5.2. IP address and port number of the next node in the sequence.
   5.3. The routing path construction certificate, generated by the anonymized node in the previous step.
   5.4. A random 128-bit ID, associated with the connection from the previous node.
   5.5. A random 128-bit ID, associated with the connection to the next node.
   5.6. The communication certificate of the next and previous node.
   5.7. A constant number of tuples containing a 128-bit seed, a size, an index and flags for creation of dummy setup packages (more info about this in the details later).
   5.8. A constant number of 128-bit seeds for stream encryption key generation + the number of keys to be generated.
   5.9. A collection of flags, telling if the node is an intermediate X-node, a terminating X-node or a Y-node, among other things.
   5.10. A secure cryptographic hash of the entire (encrypted) setup package array (see the next step), in the expected state upon reception from the previous node.
   5.11. A secure cryptographic hash of the (decrypted) contents of the current setup package.

6. The anonymized node arranges all the encrypted setup packages from the previous step in an array, in a completely randomized order, and sends off the array to the node at the beginning of the sequence, along with the pre-generated ID for this connection (see the contents of the setup package above).
7. The following sub-procedure is then performed by the receiving Y-node, and will also be repeated by each and every node in the ordered sequence, until the array again reaches the anonymized node:
    7.1. The node iterates through each encrypted package in the array, attempting to first symmetrically decrypt it using the connection ID stated by the previous node for the incoming connection, and then asymmetrically decrypt it with its own private key. It will be able to know when it has found the right one by getting a correct hash (item 5.11 in the setup package specification above).
    7.2. The node stores the contents of its own successfully decrypted package locally.
    7.3. The node authenticates the previous node (i.e. the one it received the package array from), by matching both the expected IP address (item 5.1 in the setup package specification above) against its actual IP address, and the expected ID for the incoming connection (item 5.4 in the setup package specification above) against the ID that was actually stated by the previous node along with the setup package array. Both things should always match under normal circumstances (possible exception conditions will be discussed later).
    7.4. The node matches the hash of the entire setup package array (item 5.10 in the setup package specification above) against a locally calculated hash of the array, and they should always be equal under normal circumstances (possible exception conditions will be discussed later).
    7.5. The node interprets the flags (item 5.9 in the setup package specification above), thus seeing which role in the path building process it has been allotted.
    7.6. The node makes a decision whether it is possible for it to take part in the path building process (under normal circumstances the answer should be yes, which will be assumed in this process summary, possible exception conditions will be discussed later).
    7.7. The node removes its own setup package from the array.
    7.8. The node generates faked setup packages and inserts them into the array, according to given instructions (item 5.7 in the setup package specification above).
    7.9. The node connects to the given next node, first of all validating its identity by matching its expected communication certificate (item 5.6 in the setup package specification above) against the SSL certificate used by the responding node, immediately terminating the connection if the two don't match. They should always be equal under normal circumstances (possible exception conditions will be discussed later).
    7.10. The array, in its new state, is forwarded to the next node in the sequence (item 5.2 in the setup package specification above), along with the given ID for this connection (item 5.5 in the setup package specification above).
    7.11. The next node that receives the array will repeat this procedure itself, and so on, until the array has reached the last (Y-)node in the sequence, which will have its next node set to be the original anonymized node (without for that matter knowing anything about this special circumstance itself), thus closing the circle.

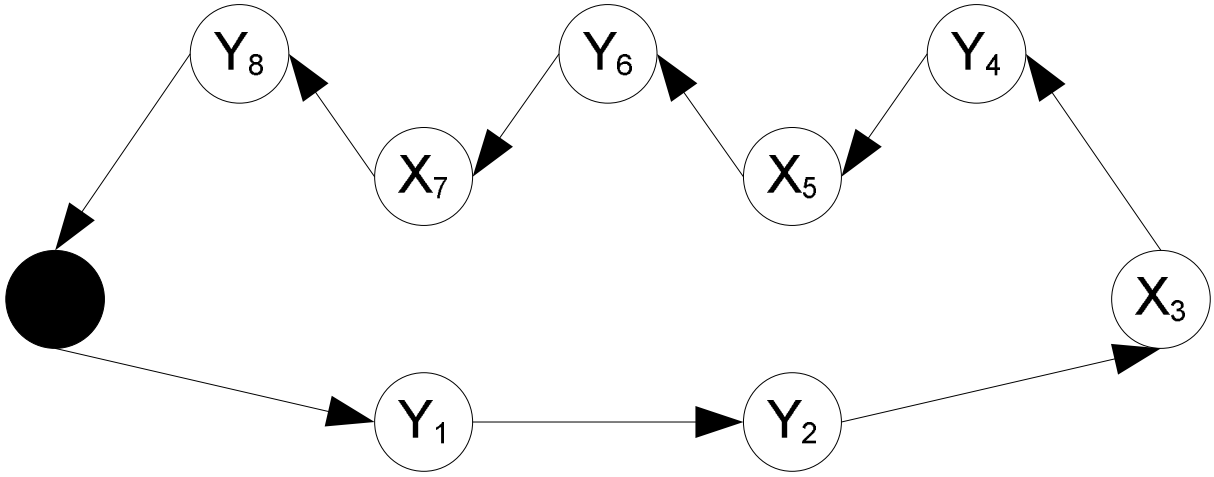

Figure 9. Routing path creation, step 7

8. If all goes well, the setup package array will traverse the entire sequence of nodes, and reach the anonymized node with a connection from the Y-node at the opposite end of the sequence, i.e. the end of the sequence that the setup package array was *not* sent into to begin with. Provided that the incoming data passes all authenticity controls (which will be discussed in more detail later), this completes the first round of the routing path setup procedure, and the anonymized node now knows that all the selected nodes accepted the routing path participation request and are standing by for the second round of the procedure, which begins with the next step.

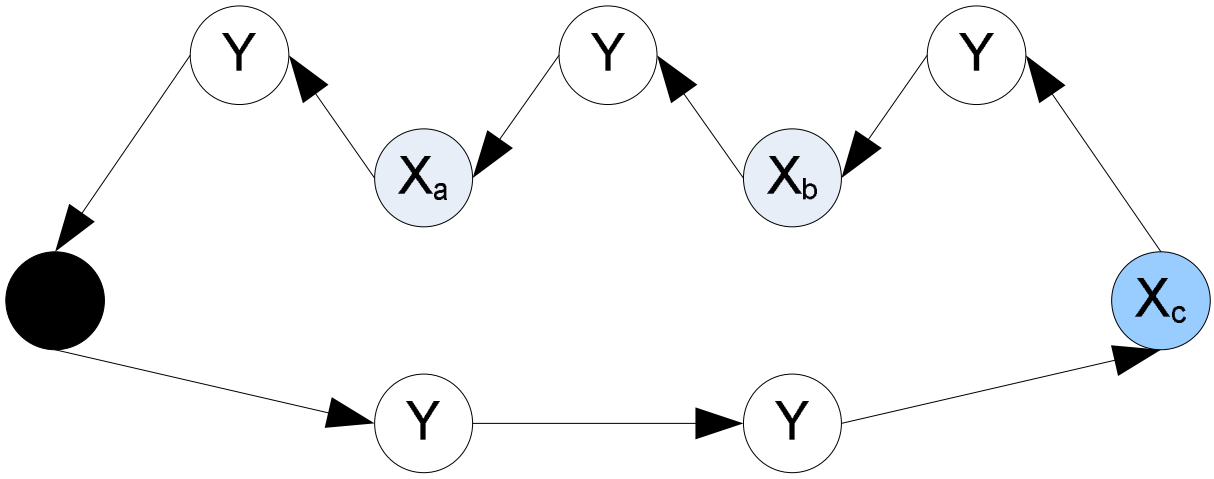

Figure 10. Routing path creation, first round completed

9. The anonymized node creates a new array of setup packages, quite similar to the first one, but with the following additions (in addition to a new random array order):

   9.1. A first round success flag is now included in the package.

   9.2. An updated set of seeds and instructions for manipulating the package array before forwarding it (equivalent to item 5.7 in the setup package specification above).

   - *For (all) X-nodes only:*

   9.3. An updated IP address of the expected new previous node in the sequence (equivalent to item 5.1 in the setup package specification above).

   9.4. An updated IP address/port for the next node in the sequence (equivalent to item 5.2 in the setup package specification above).

9.5. A random 128-bit ID, associated with the connection from the new previous node (equivalent to item 5.4 in the setup package specification above).

9.6. A random 128-bit ID, associated with the connection to the new next node (equivalent to item 5.5 in the setup package specification above).

- *For the terminating X-node only, if the path is an entry path:*

9.7. The AP address that the entry path belongs to.

9.8. A ready-made routing table entry, signed with the private key belonging to the *routing certificate* of the AP address owner (more info about this in the details later).

10. The anonymized node sends away the new array of setup packages in the same manner as the first one (and in the same direction too).

11. The following sub-procedure is then performed by each and every node in the ordered sequence, until the array again reaches the anonymized node:

11.1. The node locates, decrypts and verifies its own package in the received array, and checks the integrity of the entire array, using the same methods as in the first round.

11.2. The node now also validates the signature of the package, using the routing path construction certificate received in the package from the first round (item 5.3 in the setup package specification above).

11.3. The node confirms that the success flag is set in the package (item 9.1 in the setup package specification above).

11.4. The node modifies the package array in the same way it did in step 7.7 + 7.8 above, only now using the updated seeds and instructions (item 9.2 in the setup package specification above).

- *For Y-nodes only:*

11.5. The Y-node forwards the array, in its new state, to the next node in the sequence.

11.6. The Y-node disconnects its forward connection, and has thus fully completed its participation in the routing path building operation, and discards all information related to it, forgets all about it, and is back in the exact state it was before it was contacted to participate in the routing path building operation to begin with.

- *For (all) X-nodes only:*

11.7. The X-node checks to see if the updated expected IP address of the previous node (item 9.3 in the setup package specification above) is the same as the IP address of the existing previous node. Under normal circumstances it should never be the same, and in this case the X-node will halt and wait for an incoming connection from the expected IP address (item 9.3 in the setup package specification above) also having the right ID (item 9.5 in the setup package specification above).

11.8. When a matching incoming connection is established to the X-node (i.e. from the X-node before it), all the usual array transfer, verification and modification procedures are performed.

11.9. The X-node forwards the array, in its new state, to the next node in the sequence, still being connected from the first round.

11.10. The X-node then checks to see if the updated expected IP address and port number of the next node (item 9.4 in the setup package specification above) is the same as the IP address and port of the currently connected next node. Under normal circumstances it should never be the same, and in this case the X-node will disconnect its current forward connection, and attempt to create a new forward connection to the updated IP address and port for the stated next node (item 9.4 in the setup package specification above) also using the updated ID (item 9.6 in the

setup package specification above) for this connection. Several connection attempts might be needed at this point, since the target X-node might not have started listening for this incoming connection yet.

11.11. The X-node forwards the array, in its new state, to this new next node in the sequence too.

11.12. The intermediate Y-node between the two adjacent X-nodes has now been fully disconnected, and the final state of the routing path has been established between the current X-node and the one before it in the sequence.

11.13. The X-node now finishes the procedure by generating its unique set of stream encryption keys, according to the seeds and parameters given in the initial setup package (item 5.8 in the setup package specification above). This set of keys will be used at a later time, when establishing routing tunnels over the routing path.

- *For the terminating X-node only, if the path is an entry path:*

11.14. The terminating X-node expects the previous (Y-)node to disconnect the connection immediately after having received the package array from it. If the expected IP address (item 9.3 in the setup package specification above) and the updated ID (item 9.5 in the setup package specification above) are empty (all zero), the terminating X-node also doesn't expect any new incoming connection from a previous node, since it is located at the end of the routing path, and in this case only has the multiple Y-nodes in front of it (this will only happen in half the cases, i.e. where the randomly chosen beginning of the node sequence is the one with the multiple Y-nodes).

11.15. If the created routing path is an entry path, after having done everything else that a normal intermediate X-node is supposed to do, the terminating X-node proceeds to submit the new pre-signed routing table entry (item 9.8 in the setup package specification above) for the current routing path to the AP address-indexed routing table in the global network database, for the associated AP address (item 9.7 in the setup package specification above). The entry path is now officially announced, and any user on the anonymous network can look it up in the global network database, and use it to establish an anonymized connection (i.e. anonymized for the node that created the routing path) to the anonymized node.

11.16. If the created routing path is *not* an entry path, it will skip the previous step (11.15), and will instead be waiting for requests for outgoing connections from the anonymized node that owns the routing path (thus being located at its other end). This will be discussed further in the section about establishment of routing tunnels below.

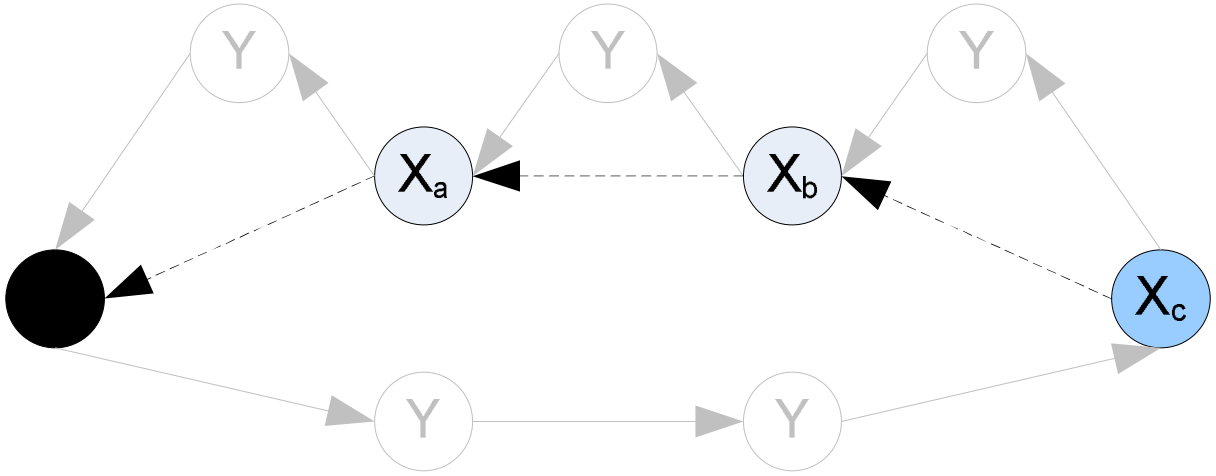

Figure 11. Routing path creation, step 11

12. If all goes well, the second setup package array will traverse the entire sequence of nodes, just like the first one, and reach the anonymized node with a connection from the Y-node (and subsequently its preceding X-node) at the end of the sequence, where it will be checked for validity. This completes the second half of the routing path setup procedure, and the routing path is now successfully and securely set up, ready for immediate use.

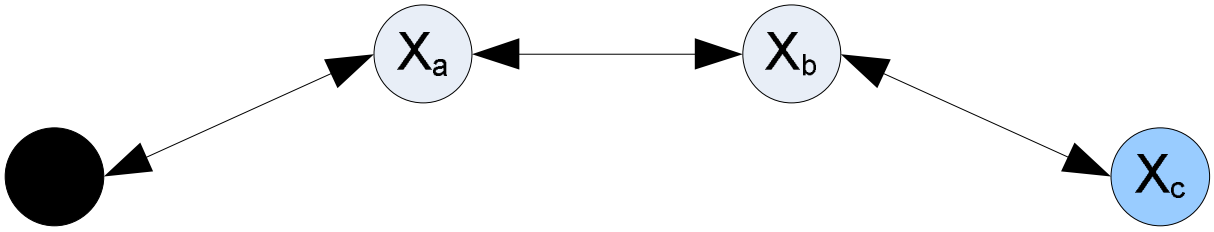

Figure 12. Routing path creation, step 12

At this point, we can clearly recognize the exact routing path structure that has been exemplified previously in this paper, and the process is complete.

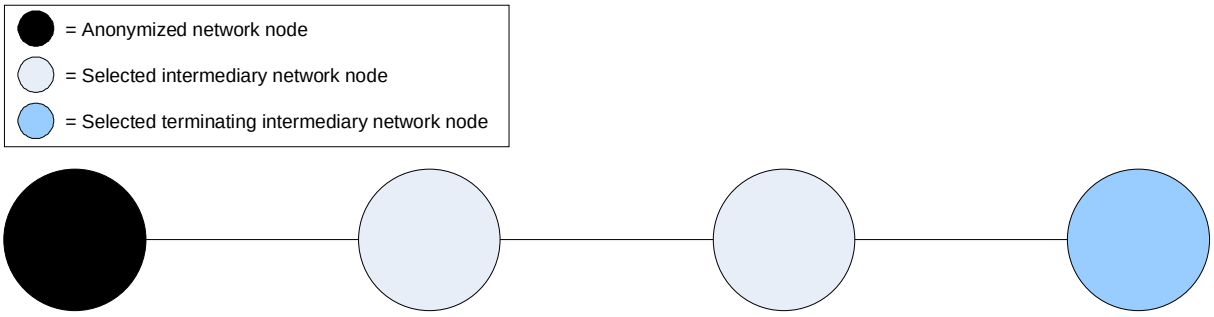

Figure 13. The completed routing path, recognizable from previously in this paper

We will now proceed with further explanation of some of the details in the process that has just been outlined, why things are done as they are, and in some cases in even more detail how they are done. After that, we will move on to explaining how routing tunnels (both of the entry and exit kind) are established over an existing routing path, and subsequently used for secure and anonymous communication.

7.1.2. Secure Establishment of Routing Paths – Low-Level Details

In this section, all the steps from the previous section (i.e. the "low level overview") will be repeated again, in grey and without the figures, and comments will be added (in normal black) to further explain the design decisions behind each step, where necessary.

1. The anonymized node (to be) selects a random set (with some exceptions to be discussed later) of x nodes from the IP address/port table of the network database, and fetches all their stored info, like e.g. their path building certificate (containing the public part of an asymmetrical encryption key-pair) and their communication certificate (containing a valid SSL certificate). These selected nodes will be called X-nodes from this point on, and are the ones that will constitute the final routing path.

First of all, about this "random set" of selected nodes, the only real requirement for randomness is that the anonymized node can be confident that no other node, or reasonably large set of nodes, has been able to influence or bias the contents of this set of nodes. Once this has been ascertained (by means of being able to trust the randomness of the returned results from the network database in this aspect, which will be discussed in more detail later), the anonymized node itself can actually make the selected set of nodes even *more* secure, by applying some extra restrictions to the nodes in the set. For example, by analyzing the IP addresses in the "initial" random set acquired from the network database, it can make sure that none of them are in the same or adjacent A, B or C nets, registered to the same ISP, or even that not too many of them are located in the same country (by using geolocation). These restrictions would reduce, to an even lower level than the initially already low one of a fully random selection, the risk of the selected set of nodes being controlled, or in any way monitored, by the same party.

Taking it even one step further, anonymized nodes could have the option to keep a list of "trusted nodes", i.e. nodes that they know are not controlled by an attacker, e.g. nodes owned by their friends etc. If just one such node is injected into each selected set of nodes, it can be completely ascertained that not all the nodes in the selected set are controlled by the same attacker. Due to how the protocol is designed, this is all that is needed to guarantee that no other node in the selected set will be able to determine the IP address of the anonymized node, even if *all* the other nodes were controlled by the same attacker. Even without these trusted nodes, however, the probability of an attacker controlling every node in a randomly selected set with the above restrictions is *extremely* low, and decreases for each new node that joins the anonymous network.

2. The anonymized node selects a similarly random set of y nodes from the IP address/port table of the network database, and fetches all their stored info, like e.g. their path building certificate, containing the public part of an asymmetrical encryption key-pair and their communication certificate (containing a valid SSL certificate). These selected nodes will be called Y-nodes from this point on.

For these Y-nodes, the same criteria and possibilities for selection of the nodes apply as for the X-nodes in step 1 above.

3. The anonymized node chooses, arbitrarily, an ordered sequence made up of all the X-nodes and Y-nodes, being constructed in a way that:
    - No two X-nodes are adjacent to each other.
    - A Y-node is located in one end of the sequence.
    - A number of Y-nodes equal to the total number of X-nodes minus one (although always at least one), are located adjacent to each other in the other end of the sequence.
    - One end of the sequence is chosen at random to be the beginning of the sequence.

This order should be chosen in a securely random way, since if there is any way to influence or predict the order in which the nodes will be placed, an attacker could possibly use this knowledge to increase the probability of a successful attack (e.g. by making sure that its nodes are always

placed first and last in all routing paths, which would in turn increase the probability of successful attacks involving massive traffic pattern correlation).

As for the underlying reasons for arranging the nodes in this manner, some further background information and explanation is required.

First of all, it needs to be understood that our main objective with these routing paths to begin with is to separate the distance between the knowledge of the IP address of the anonymized node and the knowledge of the AP address information connected to it. The concept of routing paths accomplishes this by placing the IP address knowledge in one end of a sequence of nodes, and the AP address knowledge at the other end of the same sequence of nodes (i.e. the routing path), while at the same time not allowing or making available any method for these nodes to forward any intelligible information to any other node in the sequence other than the adjacent ones (which can of course never be prevented, since adjacent nodes know the IP addresses of each other, and can thus easily always establish out-of-band communication if they were so inclined). Thus, for each intermediate node between the ends of the sequence, the probability decreases for the situation to arise where *all* the nodes in the sequence know each other, and thus will be able to know each other's identity (IP address) and be able to communicate out-of-band in order to aggregate their individual knowledge to form the complete set of knowledge, i.e. the connection between the IP address information and the AP address information.

Second of all, the reason for the "circular" communication pattern during establishment of routing paths needs to be understood. It arises from the simple fact that it enables the use of pure one-way communication between all intermediary nodes (i.e. the X and Y-nodes) during the routing path establishment process, which has two substantial primary advantages. The first is that there is an extreme increase in complexity when going from one-way communication to two way communication in any protocol. The many cases and special cases arising from increased complexity can, and will always, be exploited by attackers for their own purposes. In a protocol such as this, where it is critical that not even a single bit of information can possibly be communicated between non-adjacent nodes in the path, such an increase in complexity would be nearly fatal, and has, necessarily, been avoided at all costs. The second advantage of the circular design is that even if an attacker should somehow find a weakness in the design of the protocol, thus being able to sneak in extra data to be communicated to other, non-adjacent, nodes in the routing path, this becomes theoretically impossible without first having this data passing through the anonymized node that owns the routing path. This node will have much greater possibilities for detecting such data, and due to the current design of the protocol, it never forwards any received data in the first place.

Having established these basic design criteria for the reasons stated, we can now much better explain and motivate arranging the nodes as instructed in this step.

The reason for having "a number of Y-nodes equal to the total number of X-nodes minus one, located adjacent to each other in one end of the sequence", is because we don't want, at any point, to have a shorter distance between the anonymized node and the terminating X-node at the other end of the intended routing path than we will have in the final path (where "distance" between two nodes denotes the number of randomly selected intermediary nodes between them). These Y-nodes thus work as a temporary "buffer" to maintain this distance during the creation process of the routing path. The "total number of X-nodes minus one" will always be equal to the distance between the anonymized node and the terminating X-node in the fully established routing path, and thus this number of Y-nodes will assure that the distance, and thus the security, selected by the anonymous node to begin with, will also be maintained throughout the entire setup process of the routing path.

The reason for having "no two X-nodes adjacent to each other" and "a Y-node located in one end of the sequence" (where the other end has the multiple Y-nodes discussed above) is that no X-node should be able to know which nodes will be adjacent to it in the final routing path, and thus be able to influence, interfere or otherwise behave any differently during the path setup process based on such information (an X-node controlled by an attacker would always want to have other

X-nodes controlled by the same attacker adjacent to it, in order to achieve a fully compromised path).

Finally, the reason for "choosing one end of the sequence at random to be the beginning of the sequence" is that the direction in which the one-way communication occurs during the setup of the path should have no connection to the directions in which the anonymized node and the terminating node at the ends of the path are located. That is, the intermediary nodes should not be able to tell which direction of the path is which. Combined with the fact that a routing path can be either an entry path, an exit path, or both, the intermediary nodes will never be able to derive at which end of the path the anonymized node is located, not even by taking note of the direction in which tunnels are established over the path once it is fully setup.

4. The anonymized node generates a temporary asymmetrical cryptographic key-pair, to be used only during the setup procedure of this specific routing path. The private key of this key-pair will be called *the routing path construction key*, and the public key of this key-pair will be called *the routing path construction certificate*, from now on.

If the same routing path construction key-pair would be used for multiple routing path setup procedures, i.e. if it would *not* be regenerated each time, it could be used by an attacker, who has had his nodes selected for inclusion in more than one of them, to correlate different routing paths created by the same anonymized node. In such case, if ever only one routing path of the anonymized node were to be compromised, no matter how unlikely, the routing path construction key-pair could be used to bind all previous semi-compromised routing paths to this same anonymized node, which would be a completely unnecessary weakness.

5. The anonymized node prepares a special unique *"setup package"* for each individual X-node and Y-node, being asymmetrically encrypted with the public key of the individual recipient, symmetrically encrypted with the 128-bit ID of its incoming connection, and signed with the routing path construction key from the previous step. The package contains the following (just as with all other steps, this will be explained in more detail in the next section):

    5.1. IP address of the expected previous node in the sequence.

This will be used by each node in the routing path to verify that the node contacting them is the one intended.

    5.2. IP address and port number of the next node in the sequence.

This will be used by each node in the routing path to pass on the package array after having completed its own processing.

    5.3. The routing path construction certificate, generated by the anonymized node in the previous step.

This will be used by each node in the routing path to verify the authenticity of the package array in the second round of the routing path creation process.

    5.4. A random 128-bit ID, associated with the connection from the previous node.

This will be used by each node in the routing path to verify that the node contacting them is the one intended.

    5.5. A random 128-bit ID, associated with the connection to the next node.

This will be used by each node in the routing path to authenticate themselves to the next node in the routing path.

    5.6. The communication certificate of the next and previous node.

These SSL certificates will be used to establish two-way authenticated and securely encrypted communication between all nodes in the routing path.

    5.7. A constant number of tuples containing a 128-bit seed, a size, an index and flags for creation of dummy setup packages (more info about this in the details later).

These will be used by each node in the routing path to generate dummy packages to be inserted into the package array before it is passed on to the next node. Theses dummy packages, in turn, exist in part to prevent the possibility of any node in the routing path being able to determine how many nodes make up the path in total, and how many nodes that remain beyond it in the path before the anonymized node is reached, and in part to counteract the special case vulnerability of the first Y-node in the routing path creation sequence trying to provide an entirely fake routing path back to the anonymized node (despite not ever being able to know if its preceding node is really the anonymized node or not, but still “taking a chance”).

5.8. A constant number of 128-bit seeds for stream encryption key generation + the number of keys to be generated.

These seeds both guarantee the security of the stream encryption keys used in all tunnels being created over the routing path, and also guarantee that the anonymized node will be able to recover these keys for all such tunnels.

5.9. A collection of flags, telling if the node is an intermediate X-node, a terminating X-node or a Y-node, among other things.

5.10. A secure cryptographic hash of the entire (encrypted) setup package array (see the next step), in the expected state upon reception from the previous node.

This hash makes sure that no node can add or manipulate data inside the package array as it is being sent between the different nodes. If such a thing was possible, it would be an excellent way for adversarial nodes to communicate with other conspiring nodes throughout the routing path.

5.11. A secure cryptographic hash of the (decrypted) contents of the current setup package.

This is the hash that makes it possible for each node, once the package array arrives, to find its own package in the array. The receiving node attempts to decrypt each package in the array, and tests to see if the decryption succeeded by checking whether this hash matches the decrypted package contents.

The various contents of this package will be explained in more detail in the steps where they are individually used. The most important thing to note here is that all these items will be securely communicated to the individual recipients, in later steps. This step can principally be seen as a summary of the contents.

6. The anonymized node arranges all the encrypted setup packages from the previous step in an array, in a completely randomized order, and sends off the array to the node at the beginning of the sequence, along with the pre-generated ID for this connection (see the contents of the setup package above).

Secure randomness in the ordering of the packages in the array is important to insure that no information about the order of the different nodes in the path can be derived from the array, or even from each individual node’s own position in the array.

7. The following sub-procedure is then performed by the receiving Y-node, and will also be repeated by each and every node in the ordered sequence, until the array again reaches the anonymized node:

7.1. The node iterates through each encrypted package in the array, attempting to first symmetrically decrypt it using the connection ID stated by the previous node for the incoming connection, and then asymmetrically decrypt it with its own private key. It will be able to know when it has found the right one by getting a correct hash (item 5.11 in the setup package specification above).

The initial symmetric decryption of each package is intended to prevent the possible weakness which would occur if an attacker could share one common asymmetrical key (or a small set of such keys) between many nodes being injected into the network, and subsequently be able to detect the presence of any of these other nodes in any routing path where at least one of the

attacker's other nodes has also been selected for participation, simply by seeing the initial array and decrypting multiple packages of it.

7.2. The node stores the contents of its own successfully decrypted package locally.

7.3. The node authenticates the previous node (i.e. the one it received the package array from), by matching both the expected IP address (item 5.1 in the setup package specification above) against its actual IP address, and the expected ID for the incoming connection (item 5.4 in the setup package specification above) against the ID that was actually stated by the previous node along with the setup package array. Both things should always match under normal circumstances (possible exception conditions will be discussed later).

In the event that this test fails, it can usually only mean that some node in the routing path has tried to tamper with the process. In all such situations, the safest thing to do is to just stop and silently let the incoming connection time out, permitting the anonymized node itself to be able to respond to this timeout. The recommended and most secure course of action for the anonymized node, when getting such a routing path creation timeout, is to completely discard the current routing path, and start the process over with an entirely new set of nodes. As mentioned previously, any attempt to allow nodes to communicate (e.g. error information) backwards through the routing path tremendously increases the complexity, and, with it, the possibility for many different kinds of attacks that would substantially reduce the security of the protocol and its routing paths.

7.4. The node matches the hash of the entire setup package array (item 5.10 in the setup package specification above) against a locally calculated hash of the array, and they should always be equal under normal circumstances (possible exception conditions will be discussed later).

This step is important in order to prevent any piggy-backing of information with the array, which could otherwise be used by nodes controlled by an attacker to convey information to other conspiring nodes located after it in the setup procedure path (most likely its own IP address, for the enabling of subsequent arbitrary out-of-channel communication).

7.5. The node interprets the flags (item 5.9 in the setup package specification above), thus seeing which role in the path building process it has been allotted.

7.6. The node makes a decision whether it is possible for it to take part in the path building process (under normal circumstances the answer should be yes, which will be assumed in this process summary, possible exception conditions will be discussed later).

The only legitimate reason for "saying no" would probably be if the node in question is too heavily loaded already. Otherwise nodes could easily be DoS attacked by opening too many paths through them. On the other hand, a built-in option in the protocol to "say no" could encourage "cheat clients", who don't share their bandwidth, but still use others' bandwidth. Then again, clients that really wanted to cheat could, of course, just disconnect any such requests completely, so leaving out the option wouldn't really be an efficient solution either. Either way, the "cheat client" scenario is something of a dilemma, and should be considered further.

7.7. The node removes its own setup package from the array.

7.8. The node generates faked setup packages and inserts them into the array, according to given instructions (item 5.7 in the setup package specification above).

The reason to have nodes create and insert fake packages into the array, is first that no node should be able to derive any information about either its relative position in the array, based on how many packages are left in the array at the time it reaches the node in question, or the total size of the routing path. Second, it also counteracts the special case vulnerability of the first Y-node in the routing path creation sequence trying to provide an entirely fake routing path back to the anonymized node (despite not ever being able to know if its preceding node is really the anonymized node or not, but still "taking a chance").

7.9. The node connects to the given next node, first of all validating its identity by matching its expected communication certificate (item 5.6 in the setup package specification above) against the SSL certificate used by the responding node, immediately terminating the connection if the two don't match. They should always be equal under normal circumstances (possible exception conditions will be discussed later).

The matching of certificates makes certain that no other computer has acquired the IP address of the intended node. This could happen for example with home broadband users that share the same IP address pool through dynamic address allocation (DHCP).

7.10. The array, in its new state, is forwarded to the next node in the sequence (item 5.2 in the setup package specification above), along with the given ID for this connection (item 5.5 in the setup package specification above).

7.11. The next node that receives the array will repeat this procedure itself, and so on, until the array has reached the last (Y-)node in the sequence, which will have its next node set to be the original anonymized node (without for that matter knowing anything about this special circumstance itself), thus closing the circle.

8. If all goes well, the setup package array will traverse the entire sequence of nodes, and reach the anonymized node with a connection from the Y-node at the opposite end of the sequence, i.e. the end of the sequence that the setup package array was *not* sent into to begin with. Provided that the incoming data passes all authenticity controls (which will be discussed in more detail later), this completes the first round of the routing path setup procedure, and the anonymized node now knows that all the selected nodes accepted the routing path participation request and are standing by for the second round of the procedure, which begins with the next step.

The anonymized node can easily confirm that no additional data has been injected into the array, and also that no data is missing, based on its total knowledge of what to expect and the checksums. The simple fact that the array came back, and from the correct IP address at that, is itself a cryptographic proof that it traversed the intended path, and only the intended path, since unauthorized access to the full set of information that would be required to fake this could only be acquired by breaking the strong encryption protecting the setup package array.

9. The anonymized node creates a new array of setup packages, quite similar to the first one, but with the following additions (in addition to a new random array order):

9.1. A first round success flag is now included in the package.

9.2. An updated set of seeds and instructions for manipulating the package array before forwarding it (equivalent to item 5.7 in the setup package specification above).

- *For (all) X-nodes only:*

9.3. An updated IP address of the expected new previous node in the sequence (equivalent to item 5.1 in the setup package specification above).

9.4. An updated IP address/port for the next node in the sequence (equivalent to item 5.2 in the setup package specification above).

9.5. A random 128-bit ID, associated with the connection from the new previous node (equivalent to item 5.4 in the setup package specification above).

9.6. A random 128-bit ID, associated with the connection to the new next node (equivalent to item 5.5 in the setup package specification above).

- *For the terminating X-node only, if the path is an entry path:*

9.7. The AP address that the entry path belongs to.

9.8. A ready-made routing table entry, signed with the private key belonging to the *routing certificate* of the AP address owner (more info about this in the details later).

The various contents of this package will be explained in each individual step where they are used. The important thing to note at this point is that all these items will be securely communicated to the individual recipients, in later steps. This step can principally be seen as a summary of the contents.

10. The anonymized node sends away the new array of setup packages in the same manner as the first one (and in the same direction too).
11. The following sub-procedure is then performed by each and every node in the ordered sequence, until the array again reaches the anonymized node:
    11.1. The node locates, decrypts and verifies its own package in the received array, and checks the integrity of the entire array, using the same methods as in the first round.
    11.2. The node now also validates the signature of the package, using the routing path construction certificate received in the package from the first round (item 5.3 in the setup package specification above).
    11.3. The node confirms that the success flag is set in the package (item 9.1 in the setup package specification above).
    11.4. The node modifies the package array in the same way it did in step 7.7 + 7.8 above, only now using the updated seeds and instructions (item 9.2 in the setup package specification above).
    - *For Y-nodes only:*
    11.5. The Y-node forwards the array, in its new state, to the next node in the sequence.
    11.6. The Y-node disconnects its forward connection, and has thus fully completed its participation in the routing path building operation, and discards all information related to it, forgets all about it, and is back in the exact state it was before it was contacted to participate in the routing path building operation to begin with.
    - *For (all) X-nodes only:*
    11.7. The X-node checks to see if the updated expected IP address of the previous node (item 9.3 in the setup package specification above) is the same as the IP address of the existing previous node. Under normal circumstances it should never be the same, and in this case the X-node will halt and wait for an incoming connection from the expected IP address (item 9.3 in the setup package specification above) also having the right ID (item 9.5 in the setup package specification above).
    11.8. When a matching incoming connection is established to the X-node (i.e. from the X-node before it), all the usual array transfer, verification and modification procedures are performed.
    11.9. The X-node forwards the array, in its new state, to the next node in the sequence, still being connected from the first round.
    11.10. The X-node then checks to see if the updated expected IP address and port number of the next node (item 9.4 in the setup package specification above) is the same as the IP address and port of the currently connected next node. Under normal circumstances it should never be the same, and in this case the X-node will disconnect its current forward connection, and attempt to create a new forward connection to the updated IP address and port for the stated next node (item 9.4 in the setup package specification above) also using the updated ID (item 9.6 in the setup package specification above) for this connection. Several connection attempts might be needed at this point, since the target X-node might not have started listening for this incoming connection yet.
    11.11. The X-node forwards the array, in its new state, to this new next node in the sequence too.

11.12. The intermediate Y-node between the two adjacent X-nodes has now been fully disconnected, and the final state of the routing path has been established between the current X-node and the one before it in the sequence.

11.13. The X-node now finishes the procedure by generating its unique set of stream encryption keys, according to the seeds and parameters given in the initial setup package (item 5.8 in the setup package specification above). This set of keys will be used at a later time, when establishing routing tunnels over the routing path.

- *For the terminating X-node only, if the path is an entry path:*

11.14. The terminating X-node expects the previous (Y-)node to disconnect the connection immediately after having received the package array from it. If the expected IP address (item 9.3 in the setup package specification above) and the updated ID (item 9.5 in the setup package specification above) are empty (all zero), the terminating X-node also doesn't expect any new incoming connection from a previous node, since it is located at the end of the routing path, and in this case only has the multiple Y-nodes in front of it (this will only happen in half the cases, i.e. where the randomly chosen beginning of the node sequence is the one with the multiple Y-nodes).

11.15. If the created routing path is an entry path, after having done everything else that a normal intermediate X-node is supposed to do, the terminating X-node proceeds to submit the new pre-signed routing table entry (item 9.8 in the setup package specification above) for the current routing path to the AP address-indexed routing table in the global network database, for the associated AP address (item 9.7 in the setup package specification above). The entry path is now officially announced, and any user on the anonymous network can look it up in the global network database, and use it to establish an anonymized connection (i.e. anonymized for the node that created the routing path) to the anonymized node.

The global network database will only allow updates of pre-existing routing table entries if the update is signed with the same key as the pre-existing entry, i.e. the AP address specific routing key of the anonymized node that registered the AP address to begin with. This prevents DoS-attacks in the form of injection of bad routing information from third-party sources into the global network database.

11.16. If the created routing path is *not* an entry path, it will skip the previous step (11.15), and will instead be waiting for requests for outgoing connections from the anonymized node that owns the routing path (thus being located at its other end). This will be discussed further in the section about establishment of routing tunnels below.

12. If all goes well, the second setup package array will traverse the entire sequence of nodes, just like the first one, and reach the anonymized node with a connection from the Y-node (and subsequently its preceding X-node) at the end of the sequence, where it will be checked for validity. This completes the second half of the routing path setup procedure, and the routing path is now successfully and securely set up, ready for immediate use.

This concludes the low-level details for the secure establishment of routing paths.

7.2. Routing Tunnels

As mentioned previously, the establishment of routing tunnels over pre-existing routing paths is a key process of the protocol, and an important part in providing its anonymity. As can be suspected, the exact procedure for setting up such routing tunnels in a secure way is of utmost importance for keeping the system theoretically secure. In this section, this procedure will be described in more detail. We will begin with an explanation of the process used for resolving AP addresses, which is the basis for creating and connecting routing tunnels to begin with. We will then move on to a shorter step-by-step description of the tunnel setup procedure, after which we will finally describe and explain each of the steps, and their underlying purposes and reasons, more thoroughly.

7.2.1. AP Address Resolution

Whenever a node in the anonymous network wants to contact another node in the anonymous network, this is accomplished through the use of AP addresses, as explained previously. All nodes that accept incoming connections on the AP address level (which many nodes may indeed not do at all) have an AP address registered to them in the network database.

The previous description of how routing tunnels are created and established explains how AP addresses for nodes are registered in the global network database. This section will fill in the few remaining details regarding how these AP addresses are used to get in touch with the node that originally registered them, or to be more precise, the node that owns the routing paths for which the AP address is registered.

This explanation will *not*, however, deal with the related lower-level database implementation issues, but rather focus only on the process up to the point of the database abstraction layer. Lower-level database implementation details will instead rather be dealt with in the “Network Database” section below.

So, these limitations having been defined, how then does the AP address resolution work? It is actually extremely simple. Let’s start by taking a look at a figure representing a typical situation in the anonymous network, an anonymized node (*α*) wanting to connect to a non-anonymized node (*β*). As shown previously, it will look as follows:

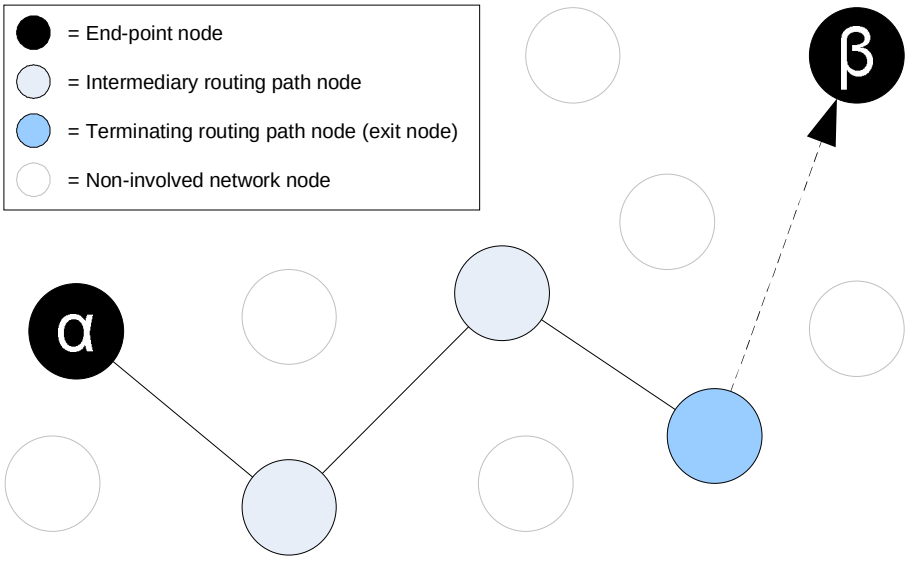

Figure 14. Anonymized node (α) wanting to connect to a non-anonymized node (β)

As can be seen in the figure, node *α* has its own routing path, in this case an exit path, which it wants to use to contact the node *β* anonymously. The details of setting up the entire connection (i.e. the routing tunnel) over the routing path will be discussed in the following section, but the

important consideration in this case is that it will be the responsibility of the terminating node of the exit path (i.e. the exit node) to resolve the AP address of *β* (after having had this AP address securely communicated to it through the tunnel from *α* as part of the routing tunnel setup process), and then establish a connection to *β*.

Once the target AP address of the desired outbound connection has been securely communicated from *α* to the exit node of its routing path (a process which, as mentioned above, will be explained in the following section, regarding secure establishment of routing tunnels), all that needs to be done by the exit node is to query the global network database about the IP address of a current entry node for the requested AP address. In the situation at hand, where the node *β* has no interest in being anonymized, it acts as its own entry node, and thus, the IP address of *β* is registered for the AP address of *β* in the network database. This IP address is returned from the database, to the inquiring exit node in the routing path of *α*, which in turn connects to this IP address and requests a connection to be opened (actually, still without being able to directly tell if the IP address belongs to *β* or to the entry node of one of its entry routing paths). Very straight forward in the end, right?

Just to avoid any uncertainties, we will also include examples for the other typical situations of AP address resolution in the anonymous network, starting out with the inverse situation of the one above, i.e. a non-anonymized node (*α*) wanting to connect to an anonymized node (*β*):

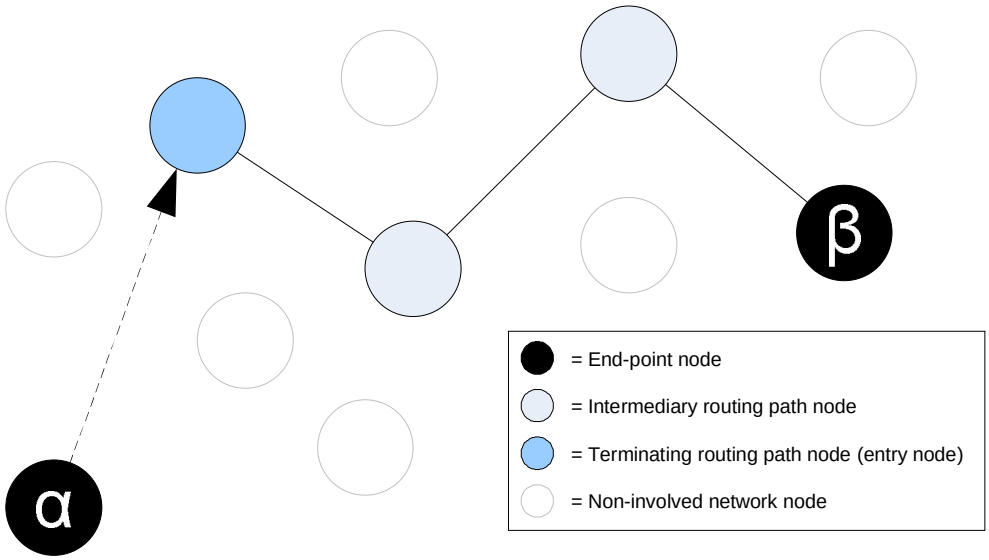

Figure 15. Non-anonymized node (α) wanting to connect to an anonymized node (β)

As can be seen in the figure, node *α* is operating on its own, thus not being anonymized, while node *β* has its own routing path, in this case an entry path, which it uses to receive connections from other nodes (in this case *α*) anonymously. In this situation, *α* acts as its own exit node, and thus has the responsibility to resolve the AP address of *β* and then establish a connection to (any valid entry node of) *β*.

So, again, all that *α* needs to do is to query the global network database for the IP address of a current valid entry node for the target AP address (i.e. the AP address of *β* in this example). Once the network database returns such an IP address as a result of this inquiry, *α* connects to this IP address (in this case the entry node of the routing path belonging to *β*), and requests a connection to be opened (likewise here, without being able to directly tell if the IP address belongs to *β* itself or to the entry node of one of its entry routing paths). It is then the responsibility of the contacted entry node to forward this request and establish a routing tunnel inside the routing path leading to *β* (a process which, as mentioned above, will be explained in the section below, regarding secure establishment of routing tunnels).

Starting to get a grip on the resolution and connection process now? Let's just include one final example of the third and last typical connection situation, i.e. an anonymized node (*α*) wanting to connect to another anonymized node (*β*):

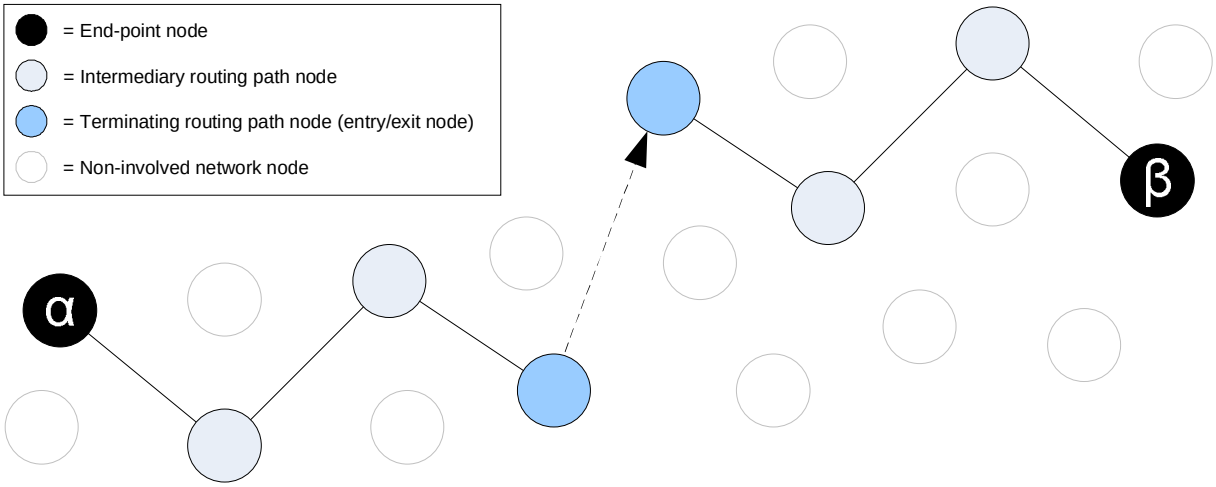

Figure 16. Anonymized node (α) wanting to connect to another anonymized node (β)

As can be seen in the figure, node *α* has its own routing path, in this case an exit path, which it wants to use to contact the node *β* anonymously. Node *β* also has its own routing path, in this case an entry path, which it uses to receive connections from other nodes (in this case *α*) anonymously. Just as in the previous examples, it will be the responsibility of the terminating node of the exit path (i.e. the exit node) of *α* to resolve the AP address of *β*, and then establish a connection to (any valid entry node of) *β*.

Once the target AP address of the desired outbound connection has been securely communicated from *α* to the exit node of its routing path, all that remains to be done by the exit node is to query the global network database about the IP address of a current entry node for the requested AP address. Once *α* gets hold of such an IP address from the network database, it connects to this IP address (in this case the entry node of the routing path belonging to *β*), and requests a connection to be opened (likewise here, without being able to directly tell if the IP address belongs to *β* itself or to the entry node of one of its entry routing paths). It is then the responsibility of the contacted entry node to forward this request and establish a routing tunnel inside the routing path leading to *β*.

This concludes the explanation of how AP addresses are resolved and used to get in touch with any node of choice within the anonymous network, and we will now move on to describing the full process used for establishing routing tunnels (i.e. new connections) over existing routing paths.

7.2.2. Secure Establishment of Routing Tunnels – Low-Level Overview

Now that we know all about how routing paths are created by all nodes that want to be anonymous, and also on a higher level how such routing paths and their respective anonymized nodes find and connect to each other by means of their AP addresses, it is time to describe the last crucial step involved in the process of creating anonymized end-to-end connections between two nodes in the anonymized network, namely, the establishment of routing tunnels.

As previously described, a routing tunnel is the logical representation of a specific connection going over a specific routing path. It should be thoroughly understood that a routing tunnel does not reach from one anonymized node to another. Rather, it only exists within the bounds of a single routing path, being fully owned and managed by only one anonymized node, i.e. the one that owns the routing path that hosts the tunnel. A full connection between two anonymized nodes in the anonymized network is made up of two such routing tunnels, being connected with a single link between the respective terminating nodes of the tunnels (i.e. one exit node and one entry node).

In this section we will present a shorter step-by-step description of the process of securely setting up a routing tunnel over a pre-existing routing path. In the next section we will then be moving on to describing and explaining each of the steps, and their underlying purposes and reasons, more thoroughly.

Depending on whether the tunnel is an inbound tunnel (i.e. initiated by the entry node of a routing path, in response to an incoming connection from a third-party) or an outbound tunnel (i.e. initiated by the anonymized node that owns the routing path, as a result of this node wanting to create a connection to another AP address in the anonymous network), the process will be internally different. Seen from the outside, or even from the viewpoint of all intermediary nodes in the affected routing path, however, the process is seemingly identical and symmetrical, which is an important property further aiding the anonymity and zero-knowledge of the system, as will be further described below.

Outbound Routing Tunnel Setup Procedure

That being said, here follows the procedure for securely setting up an *outbound* routing tunnel over a pre-existing routing path, followed by the procedure to set up its *inbound* counterpart.

1. The anonymized node owning the outbound routing path (i.e. exit path) wants to establish a connection to another node in the anonymous network.
2. The anonymized node generates a dummy random data package, having the size of a single symmetric crypto block (i.e. 128 bits or more), and sends this package off through the pre-existing connection to the next intermediate node in the routing path.
3. Each subsequent node in the routing path then performs the following sub-procedure:
    3.1. The node randomly selects one of the stream encryption keys from its local set of such, generated previously in step 11.13 of the routing *path* creation process.
    3.2. The node encrypts the dummy package (at this point already previously encrypted by all preceding nodes in the routing path), as a single block, with the chosen key.
    3.3. The node saves the encrypted dummy package in a time limited cache (containing all such recently forwarded dummy packages), together with the encryption key that was chosen for the package, and then sends off the encrypted dummy package to the next intermediate node in the routing path (i.e. its adjacent X-node if using the same nomenclature as during the creation procedure for routing paths) over the already established connection remaining since the routing path setup process.

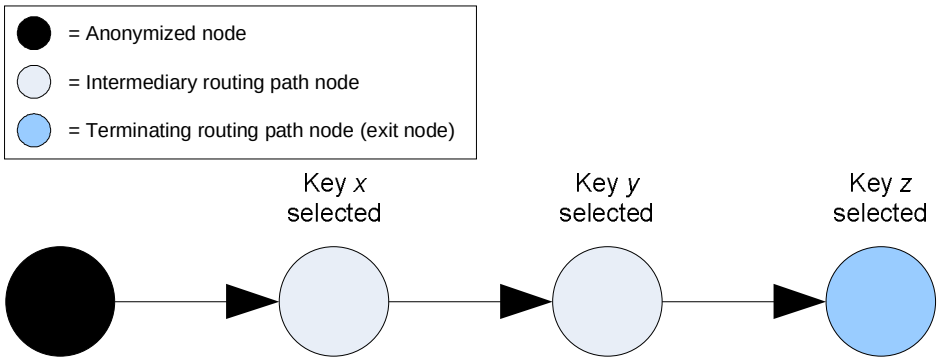

Figure 17. Outbound routing tunnel creation, step 3

4. The exit node receives the dummy package after it has traversed the entire routing path, at this point having been encrypted once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of stream encryption keys.
5. The exit node prepares a special *tunnel initialization package*, having the size of a single symmetric crypto block (i.e. 128 bits or more), containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    5.1. A *crypto key initialization block*, constructed in such a way that the risk of falsely diagnosing it as a positive in a certain kind of test will be one in 2^{32}, or less.
    5.2. A checksum of the contents of the package.
6. The exit node encrypts the tunnel initialization package, as a single block, with the previously chosen stream encryption key.
7. The exit node establishes a completely new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticated with the original connection ID used between the nodes in the original routing path setup procedure. A copy of the originally received dummy package is then sent over this connection, immediately followed by the new and encrypted tunnel initialization package.
8. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    8.1. The node authenticates the incoming connection by its IP address combined with the given connection ID.
    8.2. The node matches the connection to the corresponding previously forwarded dummy package (and thus also to the right encryption key which was chosen during that forwarding procedure, described in step 3 above), by matching the first data block that arrives over the connection against the time limited cache of all such previously forwarded dummy packages. The matching entry is then removed from the cache after the encryption key for the new connection has been stored separately.
    8.3. The node decrypts both the first received data block (i.e. the copy of the originally forwarded dummy package) and the second one (i.e. the new tunnel initialization package), using the encryption key derived in the previous step.
    8.4. The node establishes a new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticating it in both directions by means of the original connection ID used between the nodes in the original routing path setup procedure, and the previously stored (during the routing path setup procedure) “server-side” SSL certificate for this previous node.
    8.5. The node sends the two decrypted data blocks (i.e. the copy of the originally forwarded dummy package and the new tunnel initialization package) over the newly established and authenticated connection, in the same order that they were received.

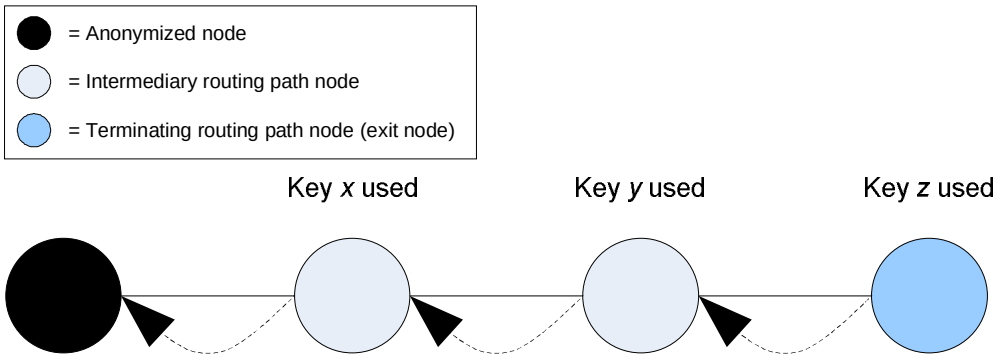

Figure 18. Outbound routing tunnel creation, step 8

9. The anonymized node finally receives the tunnel initialization package, after it has traversed the entire routing path, at this point “decrypted” (in the present circumstances effectively working rather as an encryption, but the terms are important here) once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of keys, which *are* known by the anonymized node (since it created the seeds and parameters for generating these keys, and can thus duplicate this process locally). At this point, the separate connection for the routing tunnel has also been successfully established throughout the entire routing path.
10. The anonymized node calculates which exact key out of its known set of keys that each individual node in the routing path has selected for this particular tunnel, i.e. the key that was used by each of them to decrypt the tunnel initialization package. This is done by brute forcing over all keys in all the known sets of keys for each individual node in the routing path. The number of keys in the set of each node can be arranged in a way that this entire operation takes no more than e.g. 0.1 – 0.5 seconds (or any other chosen length of time), during which still a considerable amount of e.g. AES blocks can be processed in memory. The test being performed on the resulting plaintext block to know if a certain brute force attempt was successful is related to the crypto key initialization block, mentioned in step 5.1 above (just as with all other steps, this will be explained in much more detail in the next section).
11. When all the chosen keys of each individual node in the entire routing path have been recovered, a final validation of the checksum in the recovered package is performed, reducing the risk of false positives to one in 2^{64} or less. It should always match under normal circumstances (possible exception conditions will be discussed later).
12. The anonymized node prepares a special *tunnel initialization reply package*, of the same size as the original tunnel initialization package, containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    12.1. The desired AP address and port to create an outbound connection to.
    12.2. A secure checksum.
13. The anonymized node encrypts the tunnel initialization reply package with all the individual recovered keys of the routing path nodes, in the appropriate sequence, in such a way that it will be decrypted correctly once it reaches the exit node, and sends it back through the newly created separate connection for the tunnel.
14. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    14.1. The node decrypts the (at this point already previously decrypted by all preceding nodes in the routing path) tunnel initialization reply package, with the previously chosen key for the connection, as a single block.

14.2. The node sends off the decrypted dummy package to the next intermediate node in the routing path, over the newly created separate connection for the tunnel.

15. The exit node finally receives the tunnel initialization reply package, after it has traversed the entire routing path, at this point being decrypted into the plaintext state in which it was created by the anonymized node at the other end of the routing path. It verifies the checksum, and then immediately attempts to create an outgoing connection to the stated AP address and port.

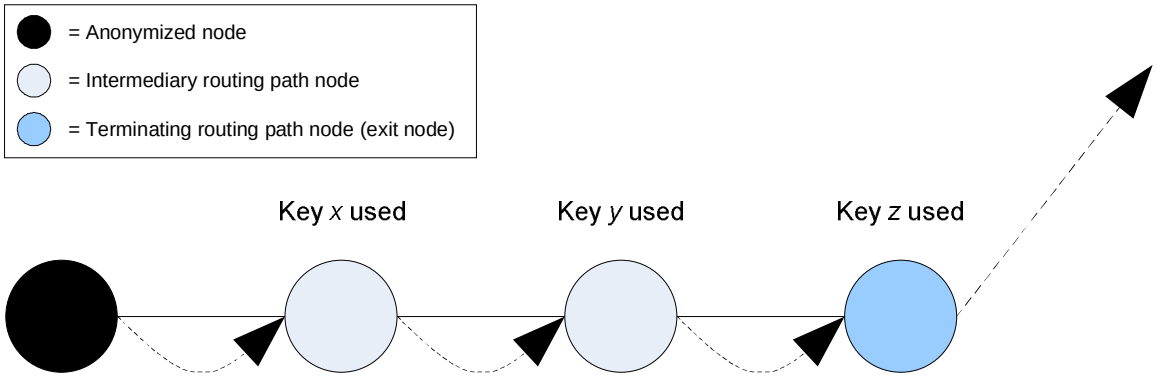

Figure 19. Outbound routing tunnel creation, step 15

16. If the external connection attempt does *not* succeed, the exit node simply closes down the newly created connection for the tunnel, which will cause a chain reaction closing down all the parts of the connection, all the way back to the anonymized node, which at this point will know that the connection attempt failed. If the external connection attempt *does* succeed, the exit node sends a dummy data package, having the same size as the tunnel initialization package and containing only random data, back through the newly established connection. Similar to the tunnel initialization package, it will be encrypted by each node with the established key for the connection, and eventually reach the anonymized node, which will then know that the connection has succeeded, and be able to start using it immediately. At this point, the routing tunnel creation is complete and the application layer on each side of the connection is notified of this, and can thus start communicating arbitrary data over the connection, equivalent to a TCP connection.

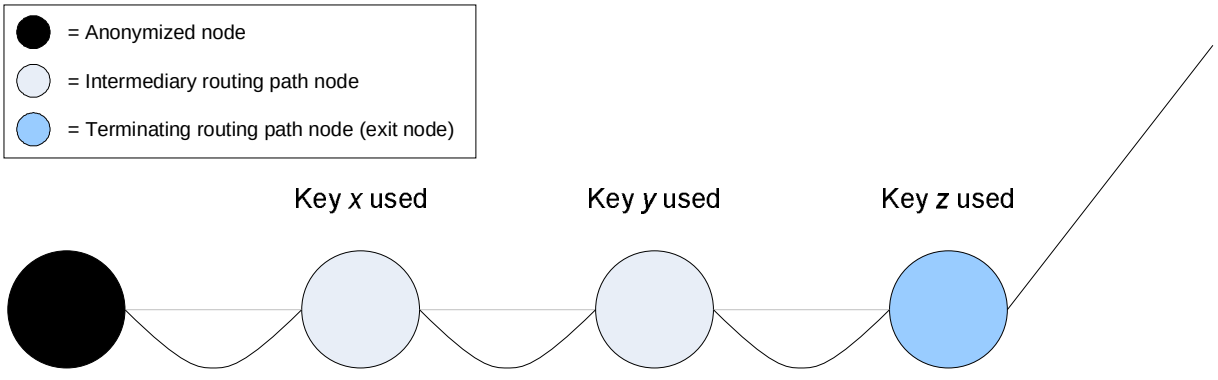

Figure 20. Outbound routing tunnel creation, completed

Inbound Routing Tunnel Setup Procedure

Having just described the procedure for securely setting up an *outbound* routing tunnel, here follows the procedure for securely setting up an *inbound* routing tunnel over a pre-existing routing path:

1. The entry node of the inbound routing path (i.e. entry path) receives a connection establishment request from another node in the anonymous network.

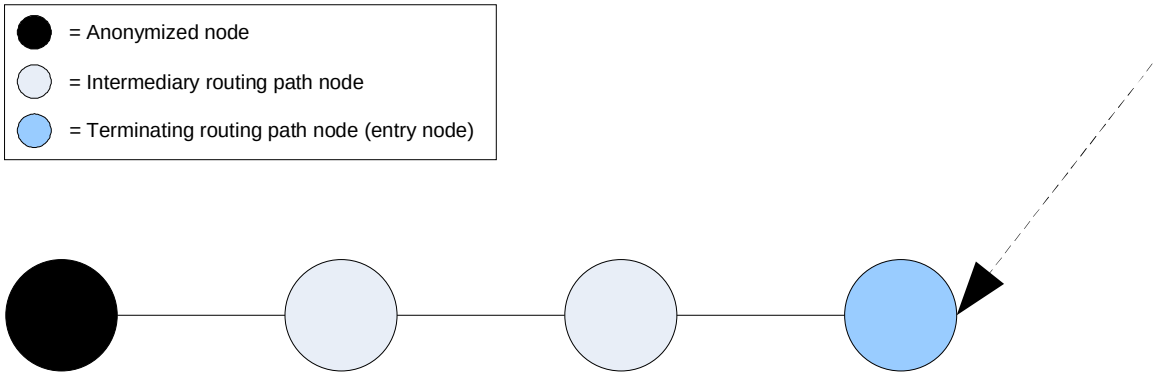

Figure 21. Inbound routing tunnel creation, step 1

2. The entry node prepares a special *tunnel initialization package*, having the same size as the initial dummy package in the outbound routing tunnel creation procedure described above, i.e. the size of a single symmetric crypto block (128 bits or more), containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    2.1. A *crypto key initialization block*, constructed in a way that the risk of falsely diagnosing it as a positive in a certain kind of test will be one in 2^{32}, or less.
    2.2. The IP address of the connecting node (Note: remember that this is only the IP address of the node that directly connects to the entry node, i.e. if the node that initiated this connection is anonymized by means of an exit path, this will only be the IP address of the exit node of this exit path, thus not revealing any sensitive information regarding the identity of the remote anonymized node).
    2.3. The AP address of the anonymized node to which the connecting node intends to create a connection.
3. The entry node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    3.1. The node randomly selects one of the stream encryption keys from its local set of such keys, which it generated previously in step 11.13 of the routing *path* creation process.
    3.2. The node encrypts the tunnel initialization package (at this point already previously encrypted by all preceding nodes in the routing path, except of course for the entry node, which doesn't have any preceding nodes and was the one to prepare the package to begin with), as a single block, with the chosen key.
    3.3. The node saves the encrypted tunnel initialization package in a time limited cache (containing all such recently forwarded tunnel initialization packages), together with the encryption key that was chosen for the package, and then sends off the encrypted tunnel initialization package to the next intermediate node in the routing path (i.e. its adjacent X-node, if using the same nomenclature as during the creation procedure for routing paths) over the already established connection remaining since the routing path setup process.

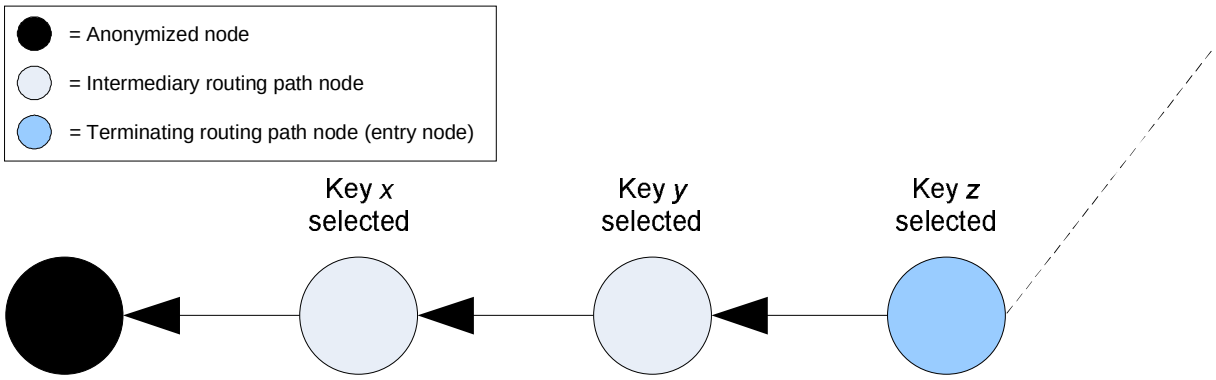

Figure 22. Inbound routing tunnel creation, step 3

4. The anonymized node that owns the routing path receives the tunnel initialization package after it has traversed the entire routing path, at this point being encrypted once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of stream encryption keys, which *are* known by the anonymized node (since it created the seeds and parameters for generating these keys, and can thus duplicate this process locally).
5. The anonymized node calculates which exact key out of its known set of keys that each individual node in the routing path has selected for this particular tunnel, i.e. the key that was used by each of them to encrypt the tunnel initialization package. This is done by brute forcing over all keys in all the known sets of keys for each individual node in the routing path. The number of keys in the set of each node can be arranged in a way that this entire operation takes no more than e.g. 0.1 – 0.5 seconds (or any other chosen length of time), during which, nevertheless, a considerable amount of e.g. AES blocks can be processed in memory. The test that is performed on the resulting plaintext block to know if a certain brute force attempt was successful is related to the crypto key initialization block, mentioned in step 2.1 above (just as with all other steps, this will be explained in much more detail in the next section).
6. When all the chosen keys of each individual node in the entire routing path have been recovered, a final validation of the target AP address of the connection is performed, to confirm that it was indeed intended for the actual AP address that the anonymized node has registered. In addition, this can be seen as an extension to the brute force key recovery procedure in the previous step, which reduces the risk of false positives to one in 2^{64}. Under normal circumstances, the AP address should always match (possible exception conditions will be discussed later).
7. The anonymized node prepares a special *tunnel initialization reply package*, of the same size as the original tunnel initialization package, containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    7.1. A flag block, containing multiple flags for communication of exceptional circumstances from the anonymized node to the entry node.
    7.2. A secure checksum.
8. The anonymized node encrypts the tunnel initialization reply package with all the individual keys of the routing path nodes, in the appropriate sequence, in such a way that it will be decrypted correctly once it reaches the entry node.
9. The anonymized node establishes a completely new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticated with the original connection ID used between the nodes in the original routing path setup procedure. A copy of the originally received tunnel initialization package is then sent over this connection, immediately followed by the new and repeatedly encrypted tunnel initialization reply package.

10. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    10.1. The node authenticates the incoming connection by its IP address combined with the given connection ID.
    10.2. The node matches the connection to the corresponding previously forwarded tunnel initialization package (and thus also the correct encryption key which was chosen during that forwarding procedure, described in step 3 above), by matching the first data block that arrives over the connection against the time limited cache of all such previously forwarded tunnel initialization packages. The matching entry is then removed from the cache after the encryption key for the new connection has been stored separately.
    10.3. The node decrypts both the first received data block (i.e. the copy of the originally forwarded tunnel initialization package) and the second one (i.e. the new tunnel initialization reply package), using the encryption key derived in the previous step.
    10.4. The node establishes a new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, and authenticates it in both directions by means of the original connection ID used between the nodes in the original routing path setup procedure, and the previously stored (during the routing path setup procedure) “server-side” SSL certificate for this previous node.
    10.5. The node sends the two decrypted data blocks (i.e. the copy of the originally forwarded tunnel initialization package, and the new tunnel initialization reply package) over the newly established and authenticated connection, in the same order that they were received.

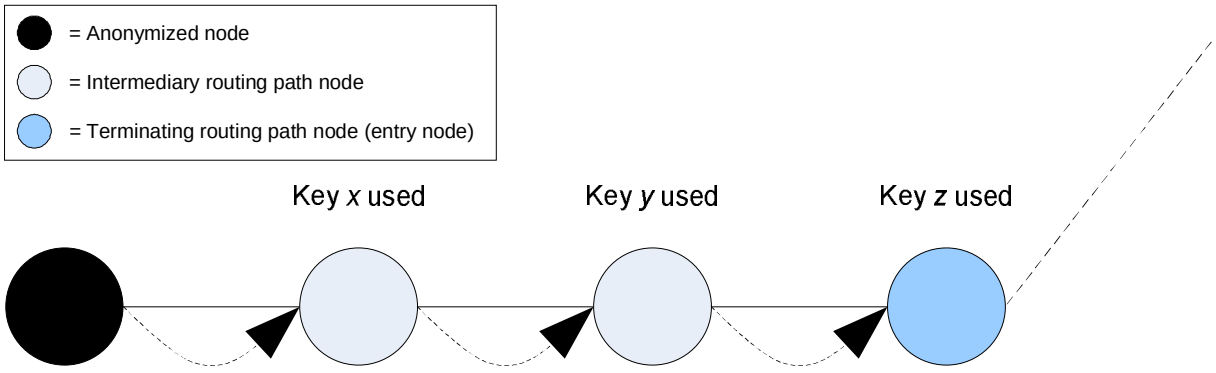

Figure 23. Inbound routing tunnel creation, step 10

11. The entry node finally receives the tunnel initialization reply package, after it has traversed the entire routing path, at this point being decrypted into the plaintext state in which it was created by the anonymized node in the other end of the routing path. The flags of the package can now be interpreted by the entry node, and under normal circumstances everything should be ok, and the routing tunnel has now been successfully established throughout the entire routing path.
12. The entry node notifies the external node that submitted the original connection establishment request that the tunnel has been set up and is ready for communication (the nature of this communication will be described in the section regarding secure communication over routing tunnels, below).

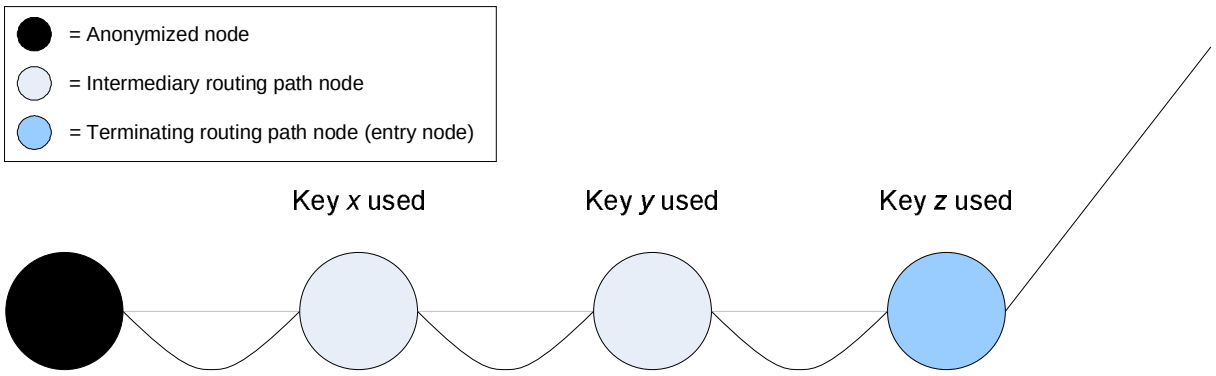

Figure 24. Inbound routing tunnel creation, completed

13. *This last step is completely unnecessary for the functionality of this procedure. It is only implemented in order to make the inbound routing tunnel setup process identical and symmetrical to the outbound routing tunnel setup process, in regards to any external observers or even any intermediary node in the routing path.*
    The entry node sends a dummy data package, having the same size as the tunnel initialization package and containing only random data, back through the newly established connection. Equivalent to the matching step of the outbound tunnel establishment procedure (step 14), it will be decrypted by each node with its established key for the connection, and eventually reach the anonymized node, which will just discard it and send a new random package of the same size back through the connection. Equivalent to the matching step of the outbound tunnel establishment procedure (step 16), it will be encrypted by each node with its established key for the connection, and eventually reach the entry node, which will then discard it. At this point, the inbound routing tunnel creation is complete and the application layer on each side of the connection is notified of this, and can thus start communicating arbitrary data over the connection, equivalent to a TCP connection.

Again, these two different procedures described above, for creation of inbound and outbound routing tunnels, are completely identical and symmetrical to any external parties. This holds true both in regards to any external eavesdropper monitoring all traffic for all the nodes in the entire routing path except the anonymized node itself and the terminating node, and also even for the intermediate X-nodes themselves, being the actual nodes constituting the tunnel. Thus, neither of these parties will be able to conclude any information about what kind of routing tunnel is being created (i.e. an inbound or outbound tunnel), or at which side of the routing path the anonymized node that owns it is located. This is, of course, yet another measure to improve the anonymity of the end-point nodes.

7.2.3. Secure Establishment of Routing Tunnels – Low-Level Details

In this section, all the steps from the previous section (i.e. the "low level overview") will be repeated again, in grey and without the figures, and comments will be added (in normal black) to further explain the design decisions behind each step, where necessary.

Outbound Routing Tunnel Setup Procedure

1. The anonymized node owning the outbound routing path (i.e. exit path) wants to establish a connection to another node in the anonymous network.
2. The anonymized node generates a dummy random data package, having the size of a single symmetric crypto block (i.e. 128 bits or more), and sends this package off through the pre-existing connection to the next intermediate node in the routing path.

Using the size of a single crypto block is good for preventing an attacker from piggy-backing data with the block as it is being forwarded through the tunnel. In the case of a single symmetric crypto block, the entire plaintext contents of it will become completely corrupted if only a single bit of it is altered by any node in the routing path.

3. Each subsequent node in the routing path then performs the following sub-procedure:
    - 3.1. The node randomly selects one of the stream encryption keys from its local set of such, which it generated previously in step 11.13 of the routing *path* creation process.
    - 3.2. The node encrypts the (at this point already previously encrypted by all preceding nodes in the routing path) dummy package with the chosen key, as a single block.
    - 3.3. The node saves the encrypted dummy package in a time limited cache (containing all such recently forwarded dummy packages), together with the encryption key that was chosen for the package, and then sends off the encrypted dummy package to the next intermediate node in the routing path (i.e. its adjacent X-node if using the same nomenclature as during the creation procedure for routing paths) over the already established connection remaining since the routing path setup process.
4. The exit node receives the dummy package after it has traversed the entire routing path, at this point being encrypted once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of stream encryption keys.
5. The exit node prepares a special *tunnel initialization package*, having the size of a single symmetric crypto block (i.e. 128 bits or more), containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    - 5.1. A *crypto key initialization block*, constructed in such a way that the risk of falsely diagnosing it as a positive in a certain kind of test will be one in 2^{32}, or less.
    - 5.2. A checksum of the contents of the package.

The crypto key initialization block could be anything that enables it to be tested really quickly and efficiently for internal consistency of some kind. One very simple example is to have two 32-bit blocks containing the exact same data (any random number). Thus, the test for internal consistency could be performed simply by comparing the two to see if they are equal or not.

The probability of a random 64-bit crypto key initialization block passing this test would be one in 2^{32}, and the extra checksum in the tunnel initialization package will make sure to decrease the probability of false positives to one in 2^{96}. The extra checksum test will practically never have to be used in the primary quick consistency test though, only in the cases where a false positive passes the primary test, which will be too infrequently to affect the testing speed anyway. This assures very high consistency test speeds (which will be important in a later step), while still not increasing the probability of false positives as a trade-off.

6. The exit node encrypts the tunnel initialization package with the previously chosen stream encryption key, as a single block.
7. The exit node establishes a completely new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticated with the original connection ID used between the nodes in the original routing path setup procedure. A copy of the originally received dummy package is then sent over this connection, immediately followed by the new and encrypted tunnel initialization package.
8. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    8.1. The node authenticates the incoming connection by its IP address combined with the given connection ID.
    8.2. The node matches the connection to the corresponding previously forwarded dummy package (and thus also to the right encryption key which was chosen during that forwarding procedure, described in step 3 above), by matching the first data block that arrives over the connection against the time limited cache of all such previously forwarded dummy packages. The matching entry is then removed from the cache after the encryption key for the new connection has been stored separately.
    8.3. The node decrypts both the first received data block (i.e. the copy of the originally forwarded dummy package) and the second one (i.e. the new tunnel initialization package), using the encryption key derived in the previous step.
    8.4. The node establishes a new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticating it in both directions by means of the original connection ID used between the nodes in the original routing path setup procedure, and the previously stored (during the routing path setup procedure) “server-side” SSL certificate for this previous node.
    8.5. The node sends the two decrypted data blocks (i.e. the copy of the originally forwarded dummy package and the new tunnel initialization package) over the newly established and authenticated connection, in the same order that they were received.
9. The anonymized node finally receives the tunnel initialization package, after it has traversed the entire routing path, at this point “decrypted” (in the situation at hand effectively working rather as an encryption, but the terms are important here) once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of keys, which *are* known by the anonymized node (since it created the seeds and parameters for generating these keys, and can thus duplicate this process locally). At this point, the separate connection for the routing tunnel has also been successfully established throughout the entire routing path.
10. The anonymized node calculates which exact key out of its known set of keys that each individual node in the routing path has selected for this particular tunnel, i.e. the key that was used by each of them to decrypt the tunnel initialization package. This is done by brute forcing over all keys in all the known sets of keys for each individual node in the routing path. The number of keys in the set of each node can be arranged in a way that this entire operation takes no more than e.g. 0.1 – 0.5 seconds (or any other chosen length of time), during which still a considerable amount of e.g. AES blocks can be processed in memory. The test being performed on the resulting plaintext block to know if a certain brute force attempt was successful is related to the crypto key initialization block, mentioned in step 5.1 above (just as with all other steps, this will be explained in much more detail in the next section).

First of all, a few words about why different keys need to be used to begin with. Why can’t each node just have a single secret symmetrical key, instead of randomly selecting one from a pre-

defined set? The reason is that we don't want the nodes in the path to be able to communicate with each other through "covert channels", i.e. using parts of the protocol as a means of encoded communication. The multiple keys prevent non-adjacent nodes from being able to communicate with each other through repeated requests for new tunnels, and indirectly through the data contained in the tunnel initialization packages of these. Using multiple keys chosen individually by the each node during the establishment of each tunnel, there are no means for a node in the path to deterministically influence the encrypted form of the tunnel initialization package coming out of its adjacent node, and even less the remaining nodes.

So, let's talk some more about the test through which the anonymized node can determine the randomly selected keys (from the predefined sets) for each nodes. To visualize this test, it can be seen as a number of nested for-loops (the number of loops being equal to the number of nodes in the routing path), where each for-loop iterates through the entire pre-defined set of keys for its corresponding node in the routing path, and the innermost loop performs the final consistency test on the crypto key initialization block. If there were to be, let's say, four nodes in a particular routing path, and they would have respectively 10, 50, 100 and 200 keys in their individual sets, this would result in a maximum of 4 * (10 * 50 * 100 * 200) = 40,000,000 block encryption tests (where the factor 4 comes from the fact that for each tested combination, one block encryption needs to be performed for each node in the path, which in this case is four nodes). Regardless of whether this is a reasonable number or not, the number of keys in the node key sets will always be automatically adjusted by the anonymized node itself to make sure that the operation always completes in a reasonable amount of time on its hardware, e.g. 0.1 – 0.5 seconds. Also, with the ongoing and upcoming mainstream deployment of massively parallel (multi core) processors, such an operation shouldn't practically have to pose any noticeable delay at all.

The main concept of this method is that any attacker (e.g. any of the intermediary nodes in the path) would rather have to crack at least the equivalence of one full 128-bit key for each of the nodes in the path in order to get hold of the used keys, an exercise which is not feasible at all (as long as the symmetrical algorithm used, e.g. AES, isn't itself cracked, of course, but even in that event it could be easily replaced too).

11. When all the chosen keys of each individual node in the entire routing path have been recovered, a final validation of the checksum in the recovered package is performed, reducing the risk of false positives to one in 2^{64} or less. It should always match under normal circumstances (possible exception conditions will be discussed later).
12. The anonymized node prepares a special *tunnel initialization reply package*, of the same size as the original tunnel initialization package, containing the following (just as with all other steps, this will be explained in much more detail in the next section):
    12.1. The desired AP address and port to create an outbound connection to.
    12.2. A secure checksum.

Just as with the dummy random data package, discussed in step 2 above, using the size of a single crypto block is good for preventing an attacker from piggy-backing data with the block as it is being forwarded through the tunnel. In the case of a single symmetric crypto block, the entire plaintext contents of it will become completely corrupted if only a single bit of it is altered by any node in the routing path, which will also be immediately discovered by use of the internal checksum when it reaches the exit node.

13. The anonymized node encrypts the tunnel initialization reply package with all the individual recovered keys of the routing path nodes, in the appropriate sequence, in such a way that it will be decrypted correctly once it reaches the exit node, and sends it back through the newly created separate connection for the tunnel.
14. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:

14.1. The node decrypts the (at this point already previously decrypted by all preceding nodes in the routing path) tunnel initialization reply package, with the previously chosen key for the connection, as a single block.

14.2. The node sends off the decrypted dummy package to the next intermediate node in the routing path, over the newly created separate connection for the tunnel.

15. The exit node finally receives the tunnel initialization reply package, after it has traversed the entire routing path, at this point being decrypted into the plaintext state in which it was created by the anonymized node in the other end of the routing path. It verifies the checksum, and then immediately attempts to create an outgoing connection to the stated AP address and port.

16. If the external connection attempt does *not* succeed, the exit node simply closes down the newly created connection for the tunnel, which will cause a chain reaction closing down all the parts of the connection, all the way back to the anonymized node, which at this point will know that the connection attempt failed. If the external connection attempt *does* succeed, the exit node sends a dummy data package, having the same size as the tunnel initialization package and containing only random data, back through the newly established connection. Similar to the tunnel initialization package, it will be encrypted by each node with the established key for the connection, and eventually reach the anonymized node, which will then know that the connection has succeeded, and be able to start using it immediately. At this point, the routing tunnel creation is complete and the application layer on each side of the connection is notified of this, and can thus start communicating arbitrary data over the connection, equivalent to a TCP connection.

Inbound Routing Tunnel Setup Procedure

1. The entry node of the inbound routing path (i.e. entry path) receives a connection establishment request from another node in the anonymous network.

2. The entry node prepares a special *tunnel initialization package*, having the same size as the initial dummy package in the outbound routing tunnel creation procedure described above, i.e. the size of a single symmetric crypto block (128 bits or more), containing the following (just as with all other steps, this will be explained in much more detail in the next section):

   2.1. A *crypto key initialization block*, constructed in a way that the risk of falsely diagnosing it as a positive in a certain kind of test will be one in 2^{32}, or less.

   2.2. The IP address of the connecting node (Note: remember that this is only the IP address of the node that directly connects to the entry node, i.e. if the node that initiated this connection is anonymized by means of an exit path, this will only be the IP address of the exit node of this exit path, thus not revealing any sensitive information in regards to the identity of the remote anonymized node).

   2.3. The AP address of the anonymized node that the connecting node intends to create a connection to by means of this connection.

The crypto key initialization block is constructed in exactly the same way as described in the comment for step 5 in the outbound routing tunnel setup procedure above. Instead of an extra checksum however, the false positive rate for the consistency test can be reduced to at least one in 2^{64} by validating that the stated AP address matches the AP address of the anonymized node.

Just as with the data packages discussed in the outbound routing tunnel setup procedure above, using the size of a single crypto block is good for preventing an attacker from piggy-backing data with the block as it is being forwarded through the tunnel.

3. The entry node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:

3.1. The node randomly selects one of the stream encryption keys from its local set of such keys, which it generated previously in step 11.13 of the routing *path* creation process.

3.2. The node encrypts the (at this point already previously encrypted by all preceding nodes in the routing path, except of course for the entry node, which doesn't have any preceding nodes and was the one to prepare the package to begin with) tunnel initialization package with the chosen key, as a single block.

3.3. The node saves the encrypted tunnel initialization package in a time limited cache (containing all such recently forwarded tunnel initialization packages), together with the encryption key that was chosen for the package, and then sends off the encrypted tunnel initialization package to the next intermediate node in the routing path (i.e. its adjacent X-node if using the same nomenclature as during the creation procedure for routing paths) over the already established connection remaining since the routing path setup process.

4. The anonymized node that owns the routing path receives the tunnel initialization package after it has traversed the entire routing path, at this point being encrypted once by each of its intermediate nodes, with randomly selected keys from each of their individual sets of stream encryption keys, which *are* known by the anonymized node (since it created the seeds and parameters for generating these keys, and can thus duplicate this process locally).

5. The anonymized node calculates which exact key out of its known set of keys that each individual node in the routing path has selected for this particular tunnel, i.e. the key that was used by each of them to encrypt the tunnel initialization package. This is done by brute forcing over all keys in all the known sets of keys for each individual node in the routing path. The number of keys in the set of each node can be arranged in a way that this entire operation takes no more than e.g. 0.1 – 0.5 seconds (or any other chosen length of time), during which still a considerable amount of e.g. AES blocks can be processed in memory. The test that is performed on the resulting plaintext block to know if a certain brute force attempt was successful is related to the crypto key initialization block, mentioned in step 2.1 above (just as with all other steps, this will be explained in much more detail in the next section).

This key derivation procedure is performed in the exact same way as described in the comment for step 10 in the outbound routing tunnel setup procedure above.

6. When all the chosen keys of each individual node in the entire routing path have been recovered, a final validation of the target AP address of the connection is performed, to confirm that it was indeed intended for the actual AP address that the anonymized node has registered. In addition, this can be seen as an extension to the brute force key recovery procedure in the previous step, which reduces the risk of false positives to one in 2^{64}. The AP address should always match under normal circumstances (possible exception conditions will be discussed later).

7. The anonymized node prepares a special *tunnel initialization reply package*, of the same size as the original tunnel initialization package, containing the following (just as with all other steps, this will be explained in much more detail in the next section):

   7.1. A flag block, containing multiple flags for communication of exceptional circumstances from the anonymized node to the entry node.

   7.2. A secure checksum.

8. The anonymized node encrypts the tunnel initialization reply package with all the individual keys of the routing path nodes, in the appropriate sequence, in such a way that it will be decrypted correctly once it reaches the entry node.

9. The anonymized node establishes a completely new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, authenticated with the original connection ID used between the nodes in the

original routing path setup procedure. A copy of the originally received tunnel initialization package is then sent over this connection, immediately followed by the new and repeatedly encrypted tunnel initialization reply package.

10. The receiving X-node performs the following sub-procedure, which is then repeated by all subsequent nodes in the routing path:
    10.1. The node authenticates the incoming connection by its IP address combined with the given connection ID.
    10.2. The node matches the connection to the corresponding previously forwarded tunnel initialization package (and thus also the right encryption key which was chosen during that forwarding procedure, described in step 3 above), by matching the first data block that arrives over the connection against the time limited cache of all such previously forwarded tunnel initialization packages. The matching entry is then removed from the cache after the encryption key for the new connection has been stored separately.
    10.3. The node decrypts both the first received data block (i.e. the copy of the originally forwarded tunnel initialization package) and the second one (i.e. the new tunnel initialization reply package), using the encryption key derived in the previous step.
    10.4. The node establishes a new connection to the previous node in the routing path, i.e. to the same IP address and port used during the original routing path setup procedure, and authenticates it in both directions by means of the original connection ID used between the nodes in the original routing path setup procedure, and the previously stored (during the routing path setup procedure) “server-side” SSL certificate for this previous node.
    10.5. The node sends the two decrypted data blocks (i.e. the copy of the originally forwarded tunnel initialization package, and the new tunnel initialization reply package) over the newly established and authenticated connection, in the same order that they were received.
11. The entry node finally receives the tunnel initialization reply package, after it has traversed the entire routing path, at this point being decrypted into the plaintext state in which it was created by the anonymized node in the other end of the routing path. The flags of the package can now be interpreted by the entry node, and under normal circumstances everything should be ok, and the routing tunnel has now been successfully established throughout the entire routing path.
12. The entry node notifies the external node that submitted the original connection establishment request that the tunnel has been set up and is ready for communication (the nature of this communication will be described in the section regarding secure communication over routing tunnels, below).
13. *This last step is completely unnecessary for the functionality of this procedure. It is only implemented in order to make the inbound routing tunnel setup process identical and symmetrical to the outbound routing tunnel setup process, in regards to any external observers or even any intermediary node in the routing path.*
    The entry node sends a dummy data package, having the same size as the tunnel initialization package and containing only random data, back through the newly established connection. Equivalent to the matching step of the outbound tunnel establishment procedure (step 14), it will be decrypted by each node with its established key for the connection, and eventually reach the anonymized node, which will just discard it and send a new random package of the same size back through the connection. Equivalent to the matching step of the outbound tunnel establishment procedure (step 16), it will be encrypted by each node with its established key for the connection, and eventually reach the entry node, which will then discard it. At this point, the inbound routing tunnel creation is complete and the application layer on each side of the connection is notified of this, and can thus start communicating arbitrary data over the connection, equivalent to a TCP connection.

Again, these two different procedures described above, for creation of inbound and outbound routing tunnels, are completely identical and symmetrical to any external parties. This holds true both in regards to any external eavesdropper monitoring all traffic for all the nodes in the entire routing path except the anonymized node itself and the terminating node, and also even for the intermediate X-nodes themselves, being the actual nodes constituting the tunnel. Thus, neither of these parties will be able to conclude any information about what kind of routing tunnel is being created (i.e. an inbound or outbound tunnel), or on which side of the routing path that the anonymized node that owns it is located. This is, of course, yet another measure to improve the anonymity of the end-point nodes.

This concludes the low-level details for the secure establishment of routing tunnels.

7.2.4. Secure Communication over Routing Tunnels

Once a routing tunnel has been established, the communication over it is extremely simple. Each intermediate node (X-node) in the routing path has one connection in either direction open for the tunnel, and a stream encryption key associated with it (the one that was chosen during the tunnel establishment procedure described above). As soon as any data arrives on either of the two connections, it will be immediately stream encrypted by the node (or decrypted, depending on the direction) byte by byte, with the associated encryption key, and the encrypted (or decrypted) byte is then sent out on the matching opposite connection of the node.

This way, each node only works as a forwarding and encrypting proxy, nothing more. The anonymized node keeps track of the encryption key and stream encryption state for each of the intermediate nodes, and can, therefore, easily decrypt and encrypt the bytes that it receives and sends over the routing path.

The reason for using stream encryption in each node is to eliminate any and all possibilities for any two non-adjacent nodes in the routing tunnel/path to communicate with each other by means of any kind of data patterns sent over the routing tunnel. For each encrypted unit of data, the stream encryption state in each node is updated, which means that it will produce completely different output even if identical data is sent over the same connection twice.

The reason for individually encrypting each byte in the data stream is that it can never be safely assumed that enough data will be sent at a time to fill an entire and even crypto block, and data can of course never be withheld until enough data has arrived in this situation (which might very well never occur in many situations and with many application level protocols).

7.3. Secure End-to-End Encryption and Authentication

End-to-end security equivalent to that of SSL, i.e. using secure end-to-end encryption and bidirectional secure asymmetrical authentication, can also be easily be accomplished over the connections of the anonymous network too. The best way is most likely to simply use a regular SSL connection inside the TCP-equivalent anonymized connection that has been described. This way, the common anonymized connection would be the anonymized equivalent of a TCP connection, while the SSL-secured version would be the anonymized equivalent of an SSL/TCP connection.

Such SSL security could of course always be applied externally, by the applications themselves on the application level, but it would be a very good idea for several reasons to build it transparently into the protocol design, for all connections. This would prevent any and all eavesdropping attacks by exit nodes, entry nodes and external attackers, and make the protocol much more secure by default.

The needed certificates could also be easily integrated into the already existing network database design in an efficient manner. They could be stored in association with each AP address entry, along with the various certificates and other information already being stored there in the design having been presented this far.

Due to the inherent problems of using a standard PKI structure with certificate authorities (CA) in an anonymized network (no certificate authority will obviously be able to positively identify the owner of any AP address, since this in the foundational concept), other methods of ensuring the authenticity of end points will have to be used. The classic “web of trust” method can still be used, where one or several trusted actors (either real identities or other AP addresses) can vouch for the authenticity of a certain AP address certificate.

Also, a second solution which could be used in parallel with the “web of trust” method is to make it possible for users to manually store trusted end-point certificates for certain AP addresses locally in their computers. These will be compared to the certificates presented by the corresponding AP addresses as soon as they are being connected to, and if a mismatch would occur, a warning will be presented to the user about this. These trusted certificates would typically be acquired and stored the first time an AP address is connected to, or even acquired manually from third party sources, e.g. trusted long-time users of the AP address in question, or from trusted websites on the normal Internet.

7.4. The Network Database

7.4.1. The Simple DHT Abstraction

In order to avoid getting stuck in the more complex low-level details of how any specific DHT (Distributed Hash Table[3]) design or algorithm works, we will remove the need for all such things by creating a simple DHT abstraction that even the most basic forms of DHT designs and algorithms will be able to meet, and use this abstraction for all further discussions. That way, practically any kind of DHT algorithm can be selected and used to implement the Phantom Network Database at a later point, without affecting any of its higher level design or features. It should be noted, however, that some DTH designs may very well already inherently support some of the features that will be discussed as being placed on top of the simple DHT abstraction. In these cases, this will only be regarded as an extra advantage for selecting this particular DHT algorithm, but not as a requirement for any selected algorithm.

The following is the simple DHT abstraction that will be used, and the terms describing it:

1. A *DHT node* is one of the networked users that constitute the DHT.
    - In the case of the DHT-based Phantom Network Database, all network nodes in the entire anonymized network will be DHT nodes in the database.
2. Any DHT node can store data in the DHT, by submitting a *DHT key* with data attached to it.
    - The DHT key can be any sequence of bytes (with some defined maximum length).
    - The data attached to the DHT key can be any sequence of bytes (with some defined maximum length, normally bigger than the maximum length of the DHT keys).
3. Data can be retrieved from the DHT by any DHT node, by submitting a query for any of its existing DHT keys.
4. The DHT will be able to handle constantly departing and newly joining DHT nodes, without losing any data (or, at least, with a very low risk of doing so).
    - This is a standard feature of all DHT designs.
5. It should be possible to quickly broadcast certain messages to all DHT nodes.
    - This is a feature which most DHT designs inherently support, as part of their overlay network structure.

7.4.2. The Phantom Network Database

Having just defined the simple DHT abstraction, we can now move on to defining the more advanced distributed Phantom Network Database abstraction. This database abstraction will be built completely upon the simple DHT abstraction described above, thus having the sole requirement of being powered by any DHT design able to live up to that simple DHT abstraction.

Here follows a description of the capabilities of the Phantom Network Database (PND), and its related terminology:

1. The database should be resilient to injection of false or unauthentic data.
    - This can be solved in part by applying voting algorithms for replies, and in part by digitally signing some of its data, where suitable and possible.
2. The database should be resilient to “net splits”, i.e. attackers being able to create an isolated part of the network that they can fully control and monitor, thus being able to trap and track unsuspecting users in it (remember that the entire security of the protocol is based on attackers not being able to control a significant percentage of all nodes in the network).

[3] http://en.wikipedia.org/wiki/Distributed_hash_table

- First of all, the designs of most DHT algorithms inherently make such an attack extremely hard to accomplish on a DHT that's already up and running. This is due to the rather extreme level of random interconnection and automatic balancing between nodes that is constantly occurring.
- The most likely method to successfully lure and isolate a certain network node into a separate network would rather be to do it initially, when the victim node is getting connected to the network. Several things could be done to make also this kind of attack much more difficult.
    - First of all, once a node has been successfully connected to the real network just once, a large amount of previously known nodes could be used to "prime" each subsequent reconnection to the network. This would assure that if just one of these nodes is still legitimate, it will work as an "interconnection" between the real network and any possible false network, giving access to all the nodes in the real network and possibly even "melting together" the real network and the false network (by means of the dynamic balancing and distribution of data throughout all known nodes that is constantly going on in a DHT), thus neutralizing the false network completely.
    - As for the very first time that a node connects to the network, it is indeed important to get an "entry address" into the network (i.e. the address of any random node that is already connected to the network) from a trusted source. Given the typical exponential growth rate of a network of this kind, it should not take long until most people "know someone" who is already in the network though, and up until that point all users could easily be primed from a central trusted web site or similar. It is important to note though that such a central server will not be needed anymore as soon as the network reaches a "critical mass", which should happen very quickly.
    - Finally, given the "melting together" scenario mentioned above, no false network would even be able to survive (at least without detection) if just a single node in that entire network would know about nodes from the real network too. In order to make sure that such a thing would always happen quickly, and thus make sure that no false network could survive for any longer period of time, network nodes could be designed to recurringly "re-prime" themselves from other personal and locally defined "trusted nodes". Such a trusted node will typically be a network node belonging to a personal friend or similar, and during the re-prime procedure the two nodes that are performing it together (always in a symmetrical fashion) would share with each other a significant amount of information (e.g. IP addresses) about the current network in which they are located, and then merge this data from the other trusted node with their currently existing knowledge about the network. This way a re-primed node originally located in a false network would also immediately get access to the entire real network, and the other nodes in the false network would be reduced to being a small part of the real network instead, which is a situation that the protocol is able to handle by design. Thus, the false network will have been neutralized.

3. The database should support the notion of virtual "tables", "table fields" and "table records" for storing data.
    - This can be easily accomplished by only allowing data of certain formats to be stored in the database, where each format belongs to a different virtual table. The data of the different allowed formats could be marked based on its data type, and this will be interpreted by all network nodes in order to know to which virtual table the data belongs.
    - The notion of table fields and records can be easily accomplished by including different fields in all the different allowed kinds of data formats, thus effectively storing tuples of specific data format in each virtual table.
    - Each table record can also be submitted with an extra DHT key unique to its parent table, facilitating the subsequent retrieval of "random" records from any specific table, by simply querying for this special table key.

4. The database should be able to return random records from a specified table in a secure fashion (i.e. in a fashion such that no single node can influence more than a maximum of one record in the returned set).
    - This can be accomplished by querying multiple nodes for the same kind of data, and then, at random, picking a maximum of one data item from each contacted node. To make it even more secure, the same nodes should never, to the extent practically possible, be queried again for random data of the same kind, or even any kind of data, depending on how many different nodes are available in the network.
5. The database should be able to enforce a permission system for certain operations, e.g. allowing only the "owner" of certain data items (i.e. the entity that first inserted the data into the database) to update or remove them, but no one else, and to only allow certain queries altogether.
    - This can be accomplished by requiring cryptographic authentication for such operations, e.g. by means of requiring a valid digital signature created with the key belonging to a cryptographic certificate that is attached to the pre-existing item to be updated (or deleted).
    - In the case of allowing only certain specific queries, while disallowing all other queries, this can be a simple matter of having each node pre-filter all queries arriving to it before processing them. The rules defining what will be allowed and not allowed can be hard coded into the application itself, together with the virtual table definitions.
6. The database should be able to enforce expiry dates for certain kinds of data (e.g. table records in certain tables), outside the control of the nodes that are submitting the data.
    - This can be accomplished by including expiry time limits for records in certain tables, and having each individual node enforcing these limits locally, dropping any stored record as soon as it has passed its expiry time.
    - In a little more detail, data added to such a table would have its exact expiry time calculated locally by each node that stores the data, based on the defined expiry time limit defined in the table in question, combined with a creation time stamp that comes with the data item itself. In order to prevent "cheating", no entries having time stamps with future times will be accepted into such a table to begin with.
7. A "command channel", where centrally authorized commands can be quickly sent to all nodes participating in the network database, should be supported.
    - This can be easily accomplished by means of a combination of the DHT broadcast feature, and commands signed with an asymmetric master key whose public half is hard coded into all clients.
        - This way, central network administration, located anywhere on the network, could perform certain administrative actions in order to manually counter large and resourceful attacks against the network, e.g. globally banning certain IP addresses or removing certain entries from the network database.
        - It is important to note, however, that no such command should ever be able to affect the computers of the network nodes in any way. Because of this, no one should have to worry about the central command key being cracked, since it will at most be able to disrupt the operation of the network, but not any of its clients. And in the unlikely even that such a central command key would be cracked or otherwise compromised, it's as easy as releasing a new version of the client with a new hardcoded key inside it to completely rectify the situation.
        - Finally, if such a command feature would still induce too much paranoia among potential users of the protocol, no matter how benign it really is, it can indeed be completely left out, at least until any hypothetical large scale attack against the network actually occurs, which the build in counter measures of the protocol won't be

able to handle automatically. After all, this might not happen at all (especially judging from the (un)success rate of various attackers trying to disrupt miscellaneous controversial distributed networks on the Internet to this date).

7.4.3. The Phantom Network Database API

Having defined the capabilities of the Phantom Network Database above, we can now finally present the exact interface that Phantom nodes will be using to interact with it, e.g. the "allowed queries and operations" on it. This interface will use a subset of the full power of the Phantom Network Database abstraction, and each and every individual node will help making sure, in the best way possible, that no other node can use anything but this subset of the full capabilities, as an extra security measure.

- `RegisterMyNodeInTheNetwork(own_ip_address, communication_certificate, path_building_certificate)`
  This API will be called by each node as soon as it goes online on the anonymous network. The result will be access granted to the network database, and registration of the node's IP address in this database at the same time.

- `ReserveNewAPAddress(routing_certificate)`
  This API will be called when a node in the network wants to acquire a time limited lease of an own incoming AP address in the network. The API returns a new reserved AP address, whose routing table entries will only be able to be updated with valid signatures from the "routing certificate" supplied as a parameter to the API. The same applies for extensions of the lease. In order not to risk compromising the anonymity of the node requesting this AP address, such a request would also have to be passed through a routing path before being issued towards the network database.

- `ExtendAPAddressLease(ap_address) signed_lease_request, routing_certificate)`
  This API will be called when a node in the network wants to extend its current lease of an AP address already reserved for it. The extension will only be granted if the lease request has a valid signature created by the same routing certificate that was used for the original reservation of the AP address.

- `UpdateRoutingTableEntry(ap_address, signed_routing_entry, routing_certificate)`
  This API will be called when a node in the network wants to update the routing table entry of its AP address in the network database, to add or remove an entry path. The update will only be granted if the new routing entry has a valid signature created by the same routing certificate that was used for the original reservation of the AP address.

- `GetRandomNodeIPAddresses(noof_addresses)`
  This API will be called when a node in the network wants a list of random IP addresses for nodes in the anonymous network, e.g. for use in building a new routing path. The network will return the given number of addresses (together with their corresponding communication certificate and path building certificate), where a maximum of only one result comes from the same network database node, in order to prevent any single node from being able to influence or bias the returned set of IP addresses in any significant way.

- GetEntryNodesForAPAddress(ap_address)
  This API will be called when a node in the network wants to resolve a given AP address to the entry nodes through which it can be contacted. A list of all such entry nodes (i.e. their IP addresses and ports) is returned.

7.5. Additional Details

7.5.1. High-Availability Routing Paths

In the routing path model presented so far, it can be noted that such paths are not resilient to disappearing nodes inside the path. If only one node becomes unavailable, e.g. shuts down their computer, the entire routing path will fail, in an irreparable fashion. In some situations this is not a very critical problem, while in other it makes the protocol less useful. Thus, having access to a high-availability version of the simple routing path would be an interesting option to have.

Even though the creation of high-availability paths that maintain all the secure properties of the normal paths isn't nearly as easy as could be thought at first glance, it is indeed possible, and, in this section we will see a brief example of such a design.

The high-availability path in this example, having double redundancy for all its nodes, would look as follows:

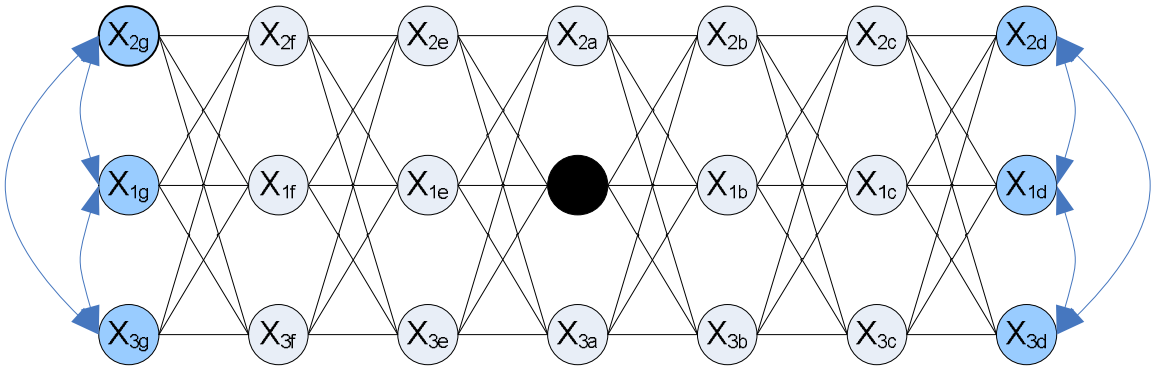

Figure 25. High-availability routing path with double redundancy.

One of the first problems encountered in this situation is that the anonymized node itself could obviously never be redundant. This causes an asymmetry, which, if not handled carefully, would make it possible for the surrounding X-nodes to know that they are adjacent to the anonymized node. Since the zero-knowledge property of the protocol in this aspect is an important part of the total strength of the anonymity it provides, we don't want to lose this property. This can be solved by making all high-availability routing paths double-ended, having the actual anonymized node located somewhere between these two ends, just as can be seen in the figure above.

As can be seen in the figure, all X-nodes now have both a number and a letter in their designation. The number represents their priority redundancy-wise, and the letter represents their position throughout the routing path.

A very short description of the workings of the high-availability routing path follows, purposely incomplete due to the large volume of details necessary to make such a routing path design work in a secure way:

- All normal communications go through the X_1 nodes, i.e. the ones in the "middle row".
- If any of these X_1 nodes should become unavailable at any point, data will be immediately routed around this new "hole" in the path by going through the corresponding X_2 node instead (and if this node should also be unavailable, the corresponding X_3 node will be used instead), without disrupting the active connection or losing data in any way.
- The communication will be immediately routed back to the main X_1 path right after this point.
- When a node malfunction such as this is detected, it will be immediately communicated to the anonymized node, through a separate command channel, which will in turn repair the routing path by replacing the lost node with a new node, using techniques similar to those used when building normal routing paths. All seamlessly while keeping the connection alive and active.

8. Legal Aspects and Implications

8.1. On a Technical Level

In order to exemplify what kind of protection the Phantom protocol really offers, and the limitations of the same, let us use one of the most extreme real-world examples of today, where people are being ferociously persecuted by powerful organizations online. Namely, file sharing.

As it now stands, in some jurisdictions a user can be sued for being connected to a torrent containing copyrighted material. Questionable evidence gathering methods aside, it might nevertheless still be claimed that the user in question willingly connected to and participated in the given torrent. These are perfectly valid grounds to sue someone today (which in practice occurs in the form of a John Doe lawsuit against the IP address in question, through which the court will later discover the true identity of the person behind this IP address, by means of a court order issued against the ISP owning the IP address).

The “next step” in this scenario might be the ability to sue someone for just using certain file sharing programs or protocols, no matter the content being accessed, or distributed, by the person in question. As far as the author of this paper knows, this has not happened yet, but the boundaries within this and similar fields are constantly being pushed forward, so it would not be overwhelmingly surprising if such legal actions were to be attempted.

Going yet a step further, generic anonymization protocols like Phantom enables people to anonymize all kinds of Internet communication, be they political discussions, communications between people in repressed regimes and people in the outside world, and also things like file sharing. This potential for “socially valuable” utility makes it much harder to altogether ban the use of such protocols, and thus, presents the opportunity for a protocol such as Phantom to facilitate and enhance the cause of free and uncensored Internet communication.

In the specific example of file sharing, it might still be claimed that people who are “unwittingly” participating in routing file sharing connections through their Phantom enabled computer could still be sued, by means of their IP addresses (i.e. the exit/entry nodes of routing paths). There are, however, some considerable differences between the current litigation models and the participation in a Phantom network. First of all, a Phantom network participant is not in any way willingly participating in the activities against which such a lawsuit is directed. Even more importantly, the user participating in the Phantom network does not in any way have access to any of the potentially “illegal” information itself, due to the encryption being employed to protect it. This makes the participant’s computer more or less equal to any router on the Internet through which an encrypted connection containing illegal information might pass, and I wouldn’t guess that any of those are to be sued anytime soon.

The next step that hypothetical Internet censors might take would be to altogether ban encryption on the Internet. This would of course make protocols like Phantom (and a very large group of other protocols) illegal, but such a turn of events would be extremely difficult to get enacted, due to the extreme consequences it would have in a wide variety of fields. Nor would it be easy to enforce. This author would rather say that it would be quite impossible to do so.

The last resort for overzealous Internet censors would be to ban the use of Internet altogether, which would of course also seem like an impossible task, considering its use and integration into the very fabric of society today.

So, the only remotely realistic, while still seemingly quite difficult, avenue of attacking the protocol and/or its users, would seem to be to attempt to sue users who unwittingly have encrypted pieces of “illegal information” passing through their computers. In order to potentially limit the effectiveness of such claims, the license of the protocol and its implementations could possibly be optimized with the goal of countering or limiting such claims, which will be discussed in the next section.

8.2. On a Legal/License Related Level

As mentioned in the previous section, the only even remotely realistic way of attempting to attack the deployment and use of the protocol (still without any possibility to track down its anonymized users though) would be to attempt to sue the users owning the entry and exit nodes of routing paths, despite the fact that these are in no way willingly sponsoring any of the communication that passes through their computers, nor have access to, or specific knowledge of, any of its contents.

In order to make such legal attacks even more difficult, it would perhaps be possible to make some adaptations to the license of the protocol and its implementations (do note that I in no way claim to have any legal skills, so this is just hopeful speculation!).

Put as simply as possible, such a license tweak would amount to stating in the license of the main protocol specification that any and all implementations of it will have to use a certain EULA.

This EULA, in turn, would state that through use of the protocol implementation in question, the user understands, and agrees, that no node in the anonymous network may be held responsible for any of the data that is being routed through it, due to the simple fact that the user neither controls what such data may contain, nor has any possibility whatsoever of accessing the data itself, and that thus, no data gathered through use of the specific protocol implementation may be used in support of a lawsuit against any of its users who are just routing data.

If such a license and EULA clause would work as intended (I'm quite sure there will be several different kinds of problems with it though, but hopefully it would contribute at least somewhat to the security of the protocol), it would put any would-be Internet censor in somewhat of a predicament, since if they use the protocol specification to create their own implementation, they will be breaking the main protocol license if they don't include the specified EULA, and if they use any legal protocol implementation, i.e. one using the specified EULA, they have been explicitly informed about the innocence of the intermediate notes of routing paths (which are the only nodes whose IP address they will ever know), and are also forbidden, by their own express agreement, from gathering data with it for any lawsuit.

Oh well, as mentioned above, just a crazy idea, which will most likely not be fully effective in all countries, but still, hopefully at least partly efficient in some countries.

In order to get it right from the start, it is hereby announced that this protocol implementation is released under this exact license.

9. Review of Design Goals

This chapter will try to assess how well the design that has been presented matches the initially established design goals. This will be done by reviewing the design goals one by one, and matching them up with features of the presented protocol design that fulfill and realize them.

9.1. Matching of Design Goals with Features of the Protocol

9.1.1. Complete Decentralization

- The protocol design has no central points, or even nodes that are individually more valuable to the collective functionality of the anonymous network than any other.
- Thus, there are no single points of the network to attack, neither technically nor legally, in order to bring down any other parts of the network than those specific ones attacked.

9.1.2. Maximum Resistance against All Kinds of DoS Attacks

- Resistance against all kinds of DoS attack vectors has been of constant concern during the design process of the protocol. It is always impossible to defend against all kinds of attacks that can in any way affect the operation of a technical solution (giant meteor smashing the earth to pieces, anyone?), and this is also an area that can always be improved infinitely (sadly in many cases at the expense of performance or resource use), but it is my sincere hope and belief that the design in its current form should make the protocol resilient enough to survive the expected and predictable kinds of attacks current on the Internet of today.

9.1.3. Theoretically Secure Anonymization

- Each and every part of the design of this protocol has been created with (theoretically) secure anonymization in mind. It is a very difficult problem for which to present a *unified* theoretical proof, and as mentioned before, some of the prerequisites and design goals forces us to use a design that is in some situations only probabilistically secure (meaning that if the same attacker owns all the randomly selected nodes in a routing path, they could theoretically connect the AP address and IP address of the anonymized node owning the same routing path, thus compromising its anonymity). That being said, the design is optimized to also minimize the probability factors of the probabilistic risks, and even make them arbitrarily small by individual user selection, with performance loss for your own communication as the trade-off. As with most non-theoretically provable security systems, however, I guess that, in the end, the truth will be decided only by the test of time (or possibly even at once by the collected hacker elite of the world, when this paper is presented in Las Vegas, but hopefully not). I can only hope that I have been able to present the different parts of the protocol in a way that makes them understandable and clear enough to enable efficient analysis and consideration.

9.1.4. Theoretically Secure End-to-End Encryption

- It is made impossible for any node in a routing tunnel or routing path to decrypt anything being sent locally inside this tunnel, by means of symmetrical stream encryption of each individual byte in all communicated data.
- SSL connections are used as an external shell for *all* connections going between nodes in the protocol, ensuring that no external attacker can eavesdrop on any data.
- SSL equivalent functionality is also suggested to always be applied *inside* each anonymized connection being made over the anonymized network (i.e. already being inside the external SSL shell and the symmetrical stream encryption provided by all routing paths).

9.1.5. Complete (Virtual) Isolation from the "Normal" Internet

- It is impossible to contact and communicate with any regular IP address on the Internet from inside the anonymous network, unless the computer at the IP address has willfully installed a Phantom client on the computer. Thus, the network cannot be used to anonymously commit illegal acts against any computer that has not itself joined the anonymous network, and thus accepted the risks involved in anonymous communication.

9.1.6. Maximum Protection against Protocol Identification/Profiling

- SSL connections are used as an external shell for *all* connections used by the protocol, and by default they also use the standard web server SSL port (tcp/443). Thus, neither the port number nor any of the contents of the communication can be directly used to distinguish it from common secure web traffic (there are of course always enough advanced traffic analysis methods to identify certain kinds of traffic, or at least distinguish traffic from a certain other kind of traffic, but if this is made sufficiently difficult, it will take up too much resources or produce too many false positives to be practically and commercially viable).

9.1.7. High Traffic Volume and Throughput Capacity

- Due to the design of the protocol, there is no practical way for a node to know if it is communicating directly with a certain node on the anonymous network, or rather with the terminating node of one of its routing paths. Thus, single point-to-point connections between two nodes on the anonymous network, without any intermediate nodes at all, can be performed while still preserving a great measure of anonymity, or at least anonymity in the form of “reasonable doubt”, which is all that is needed in many cases. This in turn will enable the transfer of very large data volumes, at speeds similar to normal, non-anonymized, Internet traffic, without for that matter using excessive resources, either from the underlying network or the participating nodes in the network.

9.1.8. Generic, Well-Abstracted and Backward Compatible Design

- The protocol emulates generic TCP communication, ready to be used for anything that common TCP communication can be used for, but in an anonymized way.
    - UDP communication could also be implemented if desired, either by simply tunneling it over the common TCP-equivalent connections, or by using an identical design based on UDP communication altogether.
- The design is also abstracted in a way that each individual part of the protocol design (e.g. the establishment of routing paths, the establishment of routing tunnels or secure end-to-end communication) can be exchanged or redesigned without the other parts being affected or having to be redesigned at the same time.
- As has been presented, the anonymization protocol can be transparently applied to any already existing networking-enabled application, by means of some simple binary hooks, without either help from the original author or having the application itself knowing anything about it.

10. Known Weaknesses

In this section, some of the known weaknesses and avenues of attacking the protocol will be presented and summarized.

1. If *all* the nodes in a routing path are being controlled by the same attacker, this attacker can synchronize the information of the individual intermediate nodes in a way that the anonymized node can be bound to the terminating intermediate node of the routing path (i.e. the entry/exit node).
    - In the case of a pure exit path, this will only result in the attacker being able to monitor which other AP addresses the anonymized node is communication with through this specific tunnel.
    - In the case of an entry path, the attacker will be able to discover the connection between the AP address of the path and the IP address of the anonymized node that owns it, and thus be able to know the identity behind this AP address in a more permanent fashion (until the anonymized node changes it's AP address anyway, if ever).
    - Under no circumstances, however, will the attacker be able to take part of any of the data being transported over the routing path, due to the end-to-end encryption that is always employed for all communications in the Phantom network.
    - One very important detail is that it will be very difficult for the attacker to conclusively know that its nodes actually constitute the entire path, because the last node in the path before the anonymized node will never be able to determine if it is actually communicating with the anonymized node itself, or with just yet another intermediate node in a routing path. This is indeed a very important strength of the protocol design.
2. If a specific attacker would be able to monitor and correlate all network traffic throughout an entire routing tunnel, this attacker would be able to trace the routing path backwards or forward from one of its endpoints to the other (by means of seeing that the exact amount of bytes that arrive to a node is being sent along to another node immediately after it has arrived to the first node, in a repeated fashion), thus successfully executing an attack with the same consequences as in item 1 above, but also in addition not having to be in doubt regarding whether the tunnel is really being terminated at the last node the attacker knows about or not.
    - Some anonymization protocols solve this problem (well, at least try to anyway), by explicitly delaying data in each routing point, and in some cases even by adding junk data connections in each step. However, since such a concept would not at all align with the Phantom design goal of being a high throughput network, the best solution is most likely rather to optimize the intermediate node selection algorithms in such a way that nodes are chosen that are not likely to be under the control and correlation capability of the same attacker, e.g. selecting the IP addresses for these nodes in such a way that they are not located at the same ISP, or not even in the same country or part of the world. This will of course also have a negative impact on throughput, so such an optimization is probably best made available as a user individual option, since this choice only affects the user's own anonymity and throughput, and nothing else.

3. Individual intermediate nodes in a routing path could try to communicate their identity (i.e. IP address) to other non-adjacent intermediate nodes throughout the same routing path, by means of different kinds of covert channels. Such covert channels would include coding information into the timing between data chunks being sent over the tunnel, or into the size of these data chunks. The only information that would need to be communicated from one attacker node to another would be their IP address, by which they could then connect directly to each other separately to exchange arbitrary amounts of information.
    - This is a real threat indeed, but again, due to the fact that none of these attacker nodes can be certain about whether it is adjacent to the anonymized node itself or not, the damage is luckily somewhat limited. And again, no actual data from the tunnel could ever be eavesdropped upon.
    - Countermeasures in the form of micro delays and data chunk size reorganization in intermediate nodes could be more or less successfully employed against these kinds of attacks, but none of these protective measures will ever be 100% secure, so this threat should indeed be taken seriously.

11. Comparison with Other Anonymization Solutions

11.1. Advantages of Phantom over TOR

Some advantages of Phantom compared to TOR are:

- Phantom is designed from the ground up with current and future anonymization needs and demand in mind.
- Phantom is compatible with all existing and future network enabled software, without any need for adaptations or upgrades.
- Phantom has much higher throughput, not being limited by a specific number of out-proxies.
    - If the “self eliminating” reasonable doubt inducing design of Phantom is taken into consideration, the maximum network throughput level is even equal to that of normal non-anonymized communication!
    - Another throughput generating design detail is the possibility of selecting your own routing nodes, thus being able to pick ones that are close to yourself on high-speed connections.
- Phantom has no maximum limit for how much data can be transferred, while TOR explicitly forbids any larger volumes of data being transferred through it.
- The Phantom network is isolated from the rest of the Internet, and thus, no participating node ever has to worry about any kind of criminal act being perpetrated against targets on the “common Internet” from their own IP address. Only nodes that have willingly joined and explicitly accepted the risks of being reachable by anonymous communication can be targets of anything bad, and thus, they also have the possibility to regulate what services they make available to the anonymous network, which even further reduces this risk.
    - Even so, any person can easily create an “out-proxy” specific to incoming traffic to their own network or servers, which would enable anonymous access only to this limited network space. Therefore, the advantages of TOR out-proxies are in no way lost, just selectively accessible, and with the possibility of much better control.
- The secrecy of all information being transferred through the Phantom network is always inherently assured, through the use of integrated and mandatory PKI and asymmetric end-to-end encryption.
    - Attacks like the “out-proxy sniffing” being possible in the TOR network are impossible with the Phantom network, due to this enforced end-to-end encryption.
- The “DNS leak” and similar kinds of attacks, that have gotten some attention in connection to the use of TOR, are made impossible by Phantom. This is because all network communication operations in the anonymized application (or even the entire operating system if so preferred) are being hooked at the operating system level. Thus, a user mode application is unable to bypass the redirection of traffic even if it wanted to, and even more so if someone attempted to trick it into doing so.
- Phantom better prevents positive identification through traffic analysis, through the exclusive use of common SSL connections for its traffic (as an “outer shell” that is). This makes it much harder to automatically block at the ISP level, since any attempts to automatically block it would most likely bring with it large amounts of false positives in the form of common legitimate encrypted web traffic (HTTPS) being blocked. ISPs would want to prevent such false positives at all costs, since web surfing stability is one of the veritable central benchmarks for how customers experience the quality of the services delivered by any ISP.

11.2. Advantages of Phantom over I2P

Some advantages of Phantom compared to I2P[4] are:

- Phantom is compatible with all existing and future network enabled software, without any need for adaptations or upgrades.
- Phantom has higher throughput, having an explicit focus on high throughput rather than low latency.
    - If the “self eliminating” reasonable doubt inducing design of Phantom is taken into consideration, the maximum throughput is even equal to normal non-anonymized communication!
    - Another throughput generating design detail is the possibility of selecting your own routing nodes, thus being able to pick ones that are close to yourself on high-speed connections.
- The secrecy of all information being transferred through the Phantom network is always inherently assured, through the use of integrated and mandatory PKI and asymmetric end-to-end encryption.
    - Since version 0.6, I2P no longer supports end-to-end encryption in all cases.
- Phantom better prevents positive identification through traffic analysis, through the exclusive use of common SSL connections for its traffic (as an “outer shell” that is). This makes it much harder to automatically block at the ISP level, since any attempts to automatically block it would most likely bring with it large amounts of false positives in the form of common legitimate encrypted web traffic (HTTPS) being blocked. ISPs would want to prevent such false positives at all costs, since web surfing stability is one of the veritable central benchmarks for how customers experience the quality of the services delivered by any ISP.

11.3. Advantages of Phantom over Anonymized File Sharing Software

There already exists some file sharing applications with built-in anonymization of different degrees and levels of quality. Without going into their specific technical properties, it can quickly be established that these are much more likely to be the victims of “general bans” of the entire protocol or application, since it is much easier to claim that “X% of all files shared through this protocol are illegal, there are also protocols with equal file transfer capability, and thus it would be justified to ban/block this protocol altogether”.

When it comes to generic anonymization protocols like Phantom, it is completely impossible to make any estimate of how large a part of its traffic is being used for any certain kind of activity, since the only way to know what is being transferred through a Phantom connection it to be one of the end-points of the communication itself, i.e. either being the origin or the intended receiver of the transferred data.

And then last, but absolutely not least, there are of course a lot more activities benefitting from anonymization than just file transfer and file sharing. So, making the anonymization protocol generic and non-application specific does certainly increase its usefulness and potential, by several orders of magnitude, just by itself.

[4] http://www.i2p2.de

12. Summary and Future of the Protocol

This white paper is in no way a complete protocol specification, far from it actually. Its main goal is rather to provide suggestions for solutions for several typical problems that are bound to arise when designing a decentralized anonymization protocol and collect them all in a comprehensive and more or less easily digested single source of documentation which could hopefully work as some kind of reference point for any discussions that may be inspired by it.

The author of this paper sees a great potential and future demand for a generic anonymization protocol that has most, or all, of the stipulated properties of the Phantom protocol. Knowing that the actual creation of a secure protocol of the proposed design, along with all things related to such an endeavor, normally requires both the knowledge, resources and review of many well qualified contributors, this paper is mainly an attempt to induce discussion, and inspire a more organized design and development project focused on creating such a protocol, not necessarily having all the exact properties and design details of the Phantom protocol at all. In order to minimize any future design and implementation effort, and in order to isolate the important questions specific to anonymity, care was also taken to build upon well-known and robust existing technologies where suitable, e.g. SSL and distributed hash tables.

The work on this project was started long before the idea of presenting it at a conference arose. Since such a presentation is perfectly aligned with the goal of maximizing the outreach and potential to inspire discussion, however, it was a given path of action once it came to mind.

Finally, it should be noted that several previous protocol designs and implementations exist that share some of the properties and features of the Phantom protocol. The author of this paper however, found it to be an interesting challenge to take one step back, to really think through which design goals would be most desired and important in such a protocol of today and the future, and then demonstrate by more or less detailed example that a protocol following these design criteria could, indeed, practically be brought together as a whole.

One current, particularly promising anonymization protocol, among those of which this author is currently aware, which is also the one sharing most properties with Phantom (although still lacking several of them, as listed in the previous section), is the I2P protocol. Readers are indeed encouraged to give it a closer look, in the hope of such examination inspiring even more fruitful discussions.

12.1. Central Project Location

A Google Code repository has been reserved for the project, which will hopefully be able to work as a central coordinating location for future design, development and implementation of the Phantom protocol and the ideas inspired by it.

It hosts a code repository, a discussion group, a wiki and a blog, and all the other common tools enabling participants to collaborate in such a shared project.

It can be found here:

http://code.google.com/p/phantom

Just as with any other community project, In order to live on and thrive, the Phantom project needs dedicated and knowledgeable people to participate and contribute. Thus, if you want to be a maintainer, developer or otherwise part of the project, please don't hesitate to join in at any time at the project site, or to contact me directly[5]!

[5] magnus.brading [at] fortego.se