Onion Routing Analysis (excerpt from an essay on Tor) by Heath Sias

The term 'onion routing' refers to a novel cryptographic technique for near-anonymous internet-based communication. Tor is synonymous with second generation onion routing. (First generation onion routing was not extensively deployed and will not be discussed further in this analysis.)

Onions are data structures which are ferried through the internet. An onion is comprised of a payload – a plaintext message (e.g. HTTP request) which would normally be sent as is – surrounded by successive layers of encryption. The method of applying and removing these layers of encryption while the payload is in transit ensures that the source of the message remains anonymous.

Communication over Tor's onion-routing network begins with a client's SOCKS-aware application, such as Tor's official Tor Browser (which, as will be discussed later, even provides encryption for the payload itself). A message is passed to the client's Tor proxy via its SOCKS interface. The proxy, in turn, builds a virtual circuit through the network of Tor routers on the internet, through which to pass the message. At the heart of Tor is the building of this virtual circuit and the subsequent encryption of the message into an onion.

Prior to considering the virtual circuit itself, it is necessary to discuss the cryptographic underpinnings of the onion routing network. Two concepts are of 'key' importance: Asymmetric (aka public-key) and symmetric (aka private-key) cryptography.

Symmetric-key cryptography has much in common with classical cryptography, the oldest form of secure communication. In these systems, each party uses the same 'key' to both encrypt and decrypt each other’s messages. In classical systems, this was often by way of a codex or lookup table. The German Enigma machine used during WWII was a sophisticated example of a classical cryptographic system. Modern symmetric-key systems use mathematical algorithms to achieve the same result. Encryption and decryption are trivial in such systems and are computationally efficient.

Such systems are also problematic. First, the key in question must somehow be shared between each party. Sending it in-the-open defeats the purpose of encryption – a third party can easily intercept the key, enabling them to decrypt any message encrypted with that key, or to send their own 'false' messages.

Some other method of delivery must be used. Traditionally, this has been done by face-to-face meeting or courier, but these methods don't work well for time-constrained interactions or over large distances. Second, assuming a key *has* been shared successfully, the secrecy of that key is under constant threat. It must be guarded at all times from prying eyes. If it is compromised and it is known that is has been compromised, another key must be devised and physically shared by the parties involved. If the key is unknowingly compromised, it is likely a worse situation for the ignorant parties than if no assumption of security existed at all.

Asymmetric, aka public-key, cryptography solves the problems of private-key cryptography, at the expense of efficiency. Such systems are founded on the concept of the one-way function, which was known to have cryptographic use as early as 1874, but which wasn't developed for secure computer-to-computer communication until the late 1970s, when the computational power of the computers made such efforts feasible.

The one-way function can be analogized to mixing paint. While it is relatively easy to create a desired color using specific quantities of the primary colors, it is nearly impossible to reduce a premixed paint into its constituent primary colors. Certain mathematical functions have this characteristic, such as the discrete exponential function, which Tor's cryptography is based on.

In the discrete exponential function, a small prime number (generally 2, 3, or 5, called the generator) 'g' is raised to a power of 'x', 'x' being an integer between 0 and 'p', a large prime number. This number is divided by p and the remainder returned. Hence, $g^x \bmod p = c$. While 'c' is easy to compute given 'g', 'x' and 'p', there is currently no algorithm which can return 'x' in a reasonable amount of time even when 'g', 'p' and 'c' are known. If 'p' has more than three hundred digits and 'x' has more than one hundred digits, even a modern supercomputer would take billions of years to yield 'x'. This is of critical importance and it is worth mentioning again. Even knowing the generator ('g'), the prime ('p') and the result of $g^x \bmod p$ ('c'), there is *no known way* to divulge 'x' given a sufficiently large 'p' and 'x'. As will soon be shown, this feature will be used in a novel way to allow two parties which have no foreknowledge of each other to share a symmetric key with little possibility of their secret being compromised. Further communication uses this symmetric-key in the same efficient method previously discussed.

Public-key cryptography does away with the need for both parties to share a key prior to initiating communication. To do this, Tor uses what is known as a Diffie-Hellman key exchange to facilitate discovery of a 'shared secret' (the value 'c' in the function above), completely in-the-open.

To illustrate the method, we name two 'official' parties, Bob and Alice, and a third 'unofficial' party, Eve. We assume public knowledge of [from the above function] 'g' and 'p', these values being agreed upon in-the-open by Bob and Alice at the start of communication. Bob also knows a value 'y', which is known to him alone. Likewise, Alice knows a value 'z', which is known to her alone. Eve then knows 'g' and 'p', but not 'y' or 'z'.

Bob and Alice both run the discrete exponential function substituting 'y' and 'z' respectively for 'x'. Bob's result is $g^y \bmod p = r$; Alice's result is $g^z \bmod p = s$. Bob then sends his result, 'r', to Alice and

Alice sends her result, 's', to Bob. Eve can see 'r' and 's', but is still missing (and will never know) 'y' or 'z'. Finally, Bob runs the function again, this time substituting 's' for 'g'. Alice does the same, substituting 'r' for 'g'. The result of each of these functions is the same number, 'c'. s^y mod p = r^z mod p = c. This result is a 'shared secret', the symmetric key both Bob and Alice will now use to conduct their secure communication. Eve is completely in the dark.

When the origin (client's Tor proxy) wishes to construct a tunnel through the Tor network, it must first communicate with a Tor router, known as an entry node. The addresses of entry nodes are published, though public-key cryptography ensures the security of each connection.

To achieve anonymity, the origin encapsulates its messages inside an 'onion'. In general, the outermost layer of the onion is encrypted for the entry node, the encryption layer 'padded' so that the onion maintains a uniform size. In this way, it is impossible to know how close an intercepted onion is to its place of origin. In the initial setup, the onion sent from the origin to the entry node is unencrypted, though the onion is still padded so that the entry node cannot distinguish the origin from another Tor router wishing to set up a connection.

The origin sends a 'create cell' message to the entry node which contains a session ID number (a random number), a request to establish a circuit, and the origin's half of the Diffie-Hellman handshake (the origin's result of the discrete exponential function as well as the generator and prime used by the origin to calculate the result). In response the entry node sends its half of the Diffie-Hellman handshake as well as a hash of the shared secret (hashed using the shared secret) for comparison/verification. If the origin's hash of the shared secret matches the entry node's, a secure connection has been established.

Tor currently uses three nodes to anonymize traffic: and entry node, relay node, and exit node, though the number of nodes which can be used is theoretically unbounded. To establish a secure connection with the relay node, the origin then sends a 'relay extend cell', which has a 'create cell' inside, to the entry node. Knowing that the network is to be extended, the entry node 'peels' off the outermost layer of encryption and forwards the 'create cell' within to the relay node. The relay node responds (to the entry node) with its half of the Diffie-Hellman handshake and hash of the shared secret, which it sends back to the origin as a 'relay extended cell'.

Note that the relay node does not share a secret directly with the entry node, but is unaware that it does not, and if the relay node's shared key is compromised, the entry node's is not. Thus, any information divulged by gaining access to a single shared key is limited to what that specific key protected. This property is termed forward secrecy and is an integral concept in anonymizing networks like Tor.

Onion Routing Analysis (excerpt from an essay on Tor) by Heath Sias

To reach the exit node and complete the tunnel, the origin sends a standard 'relay cell' to the entry node. This cell contains a 'relay extend cell' which, in turn, contains a 'create cell'. The entry node peels off the first layer and forwards the 'relay extend cell' to the relay node. The relay node sees that the network is to be extended and, peeling off the next layer of encryption, sends the 'create cell' to the next node (the exit node, as only the origin knows). The exit node responds to the relay node, which responds to the entry node, which responds to the origin.

The origin now holds the shared key for each of the three nodes in the 'chain', while each node only shares a key with the origin. The tunnel complete, normal web traffic can now flow (with minor modification).

To use as example an HTTP request, the origin constructs an onion as follows: at the onion's center is the HTTP request, its header stripped of identifying information. This 'layer' is encrypted for the *exit* node, which is encrypted for the *relay* node, which is encrypted for the *entry* node, forming an onion of encryption layers. As the request makes its way across the Tor network, each layer is peeled away by the appropriate router, until the exit node is finally in possession of the HTTP request. It then forwards this request to the destination server.

When the server's response arrives at the exit node, it creates an onion by encrypting the response with its shared key. It passes the newly formed onion to the relay node, which encrypts it with its shared key, and passes the resulting onion to the entry node. Likewise, the entry node encrypts the onion and passes it to the origin (again, unaware that the origin is, in fact, the origin). The origin then peels away all of the onion's layers until the response is revealed, and passes the message up to the host's web browser.

The Tor network itself is highly secure and effectively anonymizing, preventing all but the most pervasive attacks, but not all attacks rely on infiltration of the network itself. The plaintext message delivered by the exit node, for instance, could contain personally identifying information.