# Improving efficiency and simplicity of Tor circuit establishment and hidden services

Lasse Øverlier[1,2] and Paul Syverson[3]

[1] Norwegian Defence Research Establishment, P.B. 25, 2027 Kjeller, Norway
lasse.overlier@ffi.no, http://www.ffi.no/
[2] Gjøvik University College, P.B. 191, 2802 Gjøvik, Norway
lasse@hig.no, http://www.hig.no/
[3] Center for High Assurance Computer Systems
Naval Research Laboratory Code 5540, Washington, DC 20375
syverson@itd.nrl.navy.mil, http://chacs.nrl.navy.mil/ /
http://www.onion-router.net

**Abstract.** In this paper we demonstrate how to reduce the overhead and delay of circuit establishment in the Tor anonymizing network by using predistributed Diffie-Hellman values. We eliminate the use of RSA encryption and decryption from circuit setup, and we reduce the number of DH exponentiations vs. the current Tor circuit setup protocol while maintaining immediate forward secrecy. We also describe savings that can be obtained by precomputing during idle cycles values that can be determined before the protocol starts. We introduce the distinction of eventual vs. immediate forward secrecy and present protocols that illustrate the distinction. These protocols are even more efficient in communication and computation than the one we primarily propose, but they provide only eventual forward secrecy. We describe how to reduce the overhead and the complexity of hidden server connections by using our DH-values to implement valet nodes and eliminate the need for rendezvous points as they exist today. We also discuss the security of the new elements and an analysis of efficiency improvements.

## 1 Introduction

Since its public deployment in October 2003, the Tor [7] anonymizing network has been a huge success. It currently consists of around 900 server nodes (onion routers) scattered throughout all inhabited continents. With a weekly estimated 200.000+ users, and no down-time since launch, it is also the largest distributed anonymizing network in use. There are other anonymizing networks: JAP [2] and Freenet [5] are the most well-known implementations. In addition there exist several commercial services offering anonymity through anonymizing proxies, e.g., Relakks [21] and Anonymizer [1].

In this paper we describe new protocols for establishing circuits through Tor and for accessing hidden services over Tor that are substantially more efficient than those currently deployed. All the Tor modifications described in the paper

were motivated by our intent to simplify and reduce the overhead of using hidden services. However, as with the introduction of entry guards motivated by our previous analysis of hidden services [17], we discovered that much of our work applies to Tor circuits in general, not just those for accessing hidden services. For clarity of exposition, we have thus separated our presentation into protocol changes that apply to all Tor circuits and protocol changes that apply only to hidden services.

Our generally applicable protocols provide (1) reduced overhead and greater efficiency for the Tor network overall, (2) improved overhead and efficiency for Tor client machines, (3) examples to refine and explicate the concept of forward secrecy, and (4) most significantly, reduced load on individual server nodes. The basic idea of onion routing was to make low-latency anonymous communication feasible by adopting a circuit approach and limiting expensive public-key crypto use to circuit setup. This has been largely successful and, for most circuits over Tor, symmetric-key crypto has always dominated CPU consumption. Still, as the network grew past 100 nodes in Spring 2005 it became necessary to modify Tor to handle the public-key overhead by shifting the default rotation interval for used-circuits from one minute to ten minutes. Because Tor is a volunteer network, many of those who would like to contribute a node can only offer either unused spare machines—which are often older, slower, and have less memory—or machines that have other jobs to do, thus that can only spare computational resources for Tor if the overhead is not too great. Therefore, our techniques effectively lower the barrier to becoming a Tor node in significant ways and so encourage the network to grow.

In addition to the reduced computational requirements for circuit establishment, we describe reductions in message flows, both for basic circuit establishment and to establish circuits for communication with hidden services, where we use the new circuit construction and the valet nodes [18] extension to hidden services to make the design simpler. The currently implemented hidden service design [7] is complex and involves the building of four circuits collectively comprised of as many as twelve Tor server nodes—not including the service lookup, the client or the hidden service node. The latency of connecting to hidden services and interacting with them and the network load resulting from this complexity may have contributed to the relatively low number of hidden services deployed to date. There is also the effect simply that perceived complexity can imply reduced expectation of security and performance. However, the lower priority placed on maintaining and improving hidden services by the Tor developers, (not in general but simply relative to other aspects of Tor) no doubt also plays a role, as does the less immediate need for hidden services for the typical Tor user. Our protocol eliminates the rendezvous server as it is used today, and we reduce the number of involved nodes from twelve to six and the number of circuits from four to two in the best case scenario. In the worst case, there are three circuits comprised of nine nodes.

In section 2 we give a short overview of the history of onion routing and circuit telescoping. In section 3 we present new methods of setting up anonymous

tunnels, both as a proposal for reducing Tor overhead and to explicate forward secrecy, and in section 4 we look at some new methods of performing hidden service connections. Section 5 discusses anonymity, security, and efficiency of the new designs, and section 6 concludes.

## 2 Background

Onion routing is an approach to low-latency anonymous communication on public networks. The first two generations of onion routing used data structures comprised of layers of public-key encryptions to establish circuits and to distribute session keys to nodes along the circuit. The session keys were used, also in a layered fashion, to encrypt and decrypt the data traveling back and forth between the circuit initiator and responder. In the current generation of onion routing, Tor, circuits are established by Diffie-Hellman (DH) key exchange with each node in the circuit, each exchange being tunneled through the already established circuit and encrypted with established session keys. This technique has been called "telescoping" since its introduction in the Freedom Network [3]. Using DH provides (perfect)[4] forward secrecy (FS), meaning that, because keys are formed from exchanged messages rather than sent in encrypted form, once the session is over and the keys discarded, an adversary who stored all previous communication cannot decrypt it by somehow later obtaining a private key used to encrypt a session key.

Interestingly the original onion routing system designers considered but abandoned in the spring of 1996 [11] the option of using public Diffie-Hellman values to achieve efficiency gains in computation. Our intended design was to include the public DH-values from the originator inside the layers of circuit building onions, which were used in the first few generations of onion routing designs, and then to combine these with public DH keys (that we assume are DH-values used for generating keys). This is very similar to one of the protocols described below. Our focus was not on FS but simply to be more computationally efficient. We were certainly aware of FS and intentionally chose a protocol for securing links between onion routers that provided it, but we only pursued it with respect to outside attackers rather than against compromised network nodes as well. The idea of using DH for basic circuit building was simply another dropped design idea until work began on the Tor design, when it was picked up for the forward secrecy it provided and for freedom from the need to store onions against replay. The first description [10,19] and implementation of onion routing uses RSA public keys for distributing circuit session keys and DH-established link encryption between the server nodes. The current version of onion routing, Tor, uses both a DH key exchange and an RSA encryption/decryption for each step on the anonymizing tunnel setup. The computational advantages of using DH that we contemplated in 1996 have lain dormant until now.

[4] Forward secrecy was called 'perfect forward secrecy' when it was introduced and often still is. We will follow the convention common in cryptologic literature of referring to it simply as 'forward secrecy'.

Hidden services [7,17] have also been a part of onion routing since 1997 [11] and in their current form have been deployed on the public Tor network since 2004. They offer resistance to distributed DoS and other types of location oriented attacks. Hidden services are hidden by the network in the sense that their network locations cannot be found through access to the service, and this hiding makes the services suitable for censorship resistance, such as for dissidents or journalists publishing information accessible from anywhere. These location hidden services have been shown to have potential vulnerabilities [16,17] some of which have been addressed. Improvements to availability and QoS have been added [18], although these have also made the protocol even more complex.

In this paper we present various DH-based protocols for more efficient establishment of circuits in an onion routing network and present both efficiency improvements and simplifications to the existing hidden services protocol (in addition to those from our more efficient circuit setup protocols).

## 3 Circuit-Building Protocol Description

We assume that the functionality of the existing Tor protocol is known to the reader. Description of Tor circuit setup can be found in [6,7].

### 3.1 Overview

The central idea of all our protocols is to have certified ephemeral key exchange values at every server node inside the anonymizing network that the client uses to generate session keys for use with the nodes. These keys are used for symmetric encryption inside the created circuits. In this way it is similar to the originally contemplated use of DH in onion routing a decade ago. In all the protocols we describe we save computational overhead because there are now about half as many exponentiations per circuit established when compared to the existing Tor circuit building protocol. We present four protocols. The first provides the building blocks on which the others are based. We then consider issues of forward secrecy and message replay. We illustrate these via a succession of protocols in which communications efficiency and to some extent computational efficiency is in each following protocol traded off for improvements in forward secrecy or replay prevention, culminating in a protocol with the computational improvements we have already noted but that provides immediate forward secrecy. These ideas will be explained below.

### 3.2 Protocol description

Our new protocols use an ElGamal key agreement [8], which is also widely known as a half-certified Diffie-Hellman key exchange [15], to initialize the keys along the circuit. Construction of DH keys is computationally expensive, so it should happen as infrequently as possible. But new DH keys enable forward secrecy when both parameters are discarded, so it should happen as often as possible. But

rotation can also require an update of the public (or user) accessible information, so it should happen as infrequently as possible. Our presentation of the various protocols below is in part designed to explore these apparently conflicting needs and to illustrate ways to either make appropriate tradeoffs or to satisfy both needs at once.

Unlike the current Tor network, no RSA key is used to encrypt client-server or server-server communication. RSA keys are used only for the node to sign public information about itself. Thus, server initialization and publishing of node information is completed as in the current Tor implementation. (1) Every server node has a permanent server key pair $PS_{pub/priv}$ as before. The private key is used for signing server information, including the public key and all information published in the directory service. (2) The server creates DH parameters $DH_{x,pub/priv}$ that are to be used in forming circuits. The public values of are made part of the information published in the directory service together with the public server key. These public values need to be updated regularly. Following current practice for the onion keys (circuit-building RSA keys) in the Tor network, the default is to have a single server DH key good for one week, with the previous week's key being retained and usable to avoid synchronization problems at the time of switching to a new key. (We could add a list of public-key-exchange values valid in different periods of time, e.g. one new DH value every day, to the published and signed list of information about this node. This would permit the servers to have multiple values with different periods of validity to support both circuit setup and to be used with valet nodes in the hidden service design. On the other hand, this would clearly increase the directory overhead of Tor at a time when directory size is seen as a main cost of running Tor and when Tor developers are looking for ways to reduce the directory size and frequency of updates.)

### 3.3 Setting up the circuit

There are two main uses of circuit constructions within Tor. (1) Setting up a standard circuit, for example to reach an exit-node in order to retrieve information from outside the anonymizing network. (2) Setting up a circuit to a hidden service using special setup paths that protect the location of the accessed server as well as the location of the client. The latter will be described in section 4.

We have in our examples described the plain Diffie-Hellman based ElGamal key exchange protocol. An implementation could use an ECC (Elliptic Curve Cryptography) version to reduce overhead in communication and computing time when deriving the session keys. Besides technical questions, the many patents in this area would need to be investigated before recommending an ECC version for Tor use. We will not discuss here the cryptographic differences or advantages between these key exchange methods.

**Plain circuit setup, using circuit setup onions** The client, $C$, wants to communicate through nodes $X$ and $Y$ to node $Z$, and from there to exit the network to server $S$, just as in current Tor communication.

First the client wants to share an ephemeral encryption key with node $X$. Every accessible node has signed and published their global DH-parameters, $DH_{x,pub}$, as described above.[5] A discussion of the various methods to use for distributing DH-parameters and public keys will not be addressed in this paper: see [7,18] for this. Now, when the client wants to establish a communication channel, it creates its own ephemeral DH value pair for use with each node, e.g., $DH_{cx,priv/pub}$, for communication between $C$ and $X$. It then sends the public part to node $X$ together with the additional information encrypted with the newly constructed key.[6]

As noted above, this is essentially a half-certified ElGamal key agreement. Now one can use the client's contact with the first server node to tunnel information to the second, and to the third, and so on.

*The first protocol* uses this new key exchange to set up each extension of the tunnel quite similarly to the current handshake of Tor, except that we use the public Diffie-Hellman value for identifying the server node. The connection between the server node and its public DH value is established via the signature on node information that the client must verify. (Alternatively it could be possible to use pairing-based cryptography to set up an identity-based Diffie-Hellman scheme for circuit building obviating the need for signed certificates [13]. However, amongst other limitations, existing pairing-based schemes require a trusted server to generate and distribute private keys to all the server nodes. A threshold system can be used to reduce trust in a single entity for generating and distributing these keys with a concomitant increase in overhead. We therefore think it unlikely that any existing pairing-based scheme will be both practical and adequately trustable for deployment on the public Tor network. Nonetheless, the potential advantages of an identity-based scheme are clear. Thus it is a worthwhile research question to explore applying these schemes to onion routing networks and similar systems. An identity-based scheme may also be useful for similar but distinct existing applications or in other contexts.)

The setup packet to $X$ contains

$$DH_{cx,pub}, \{CREATE, ID_{cx}, data\}_{K_{cx}}$$

and the reply is similar to the current Tor key establishment "CREATED" messages, except that it can be encrypted with the common key $K_{xc}$ because the

[5] This value is calculated from their private DH-parameter, $DH_{x,pub} = g^{DH_{x,priv}}$, signed and retrieved by $C$.

[6] E.g. in plain Diffie-Hellman the key $K_{cx}$ is found by using $(DH_{cx,pub})^{DH_{x,priv}} = (DH_{x,pub})^{DH_{cx,priv}}$ as key material. Note that this key material is for use only in one circuit. If $C$ were to build another circuit through $X$ during the lifetime of $DH_{x,pub}$, it would use $DH_{x,pub}$, but would generate a new $DH_{cx,pub}$. Note also that $K_{cx} \neq (DH_{cx,pub})^{DH_{x,priv}}$: as is usual cryptographic practice, a key derivation function (kdf) is needed to produce $K_{cx}$ from $(DH_{cx,pub})^{DH_{x,priv}}$, and a different kdf is used for $K_{cx}$ (for communication from $C$ to $X$) than the kdf used for $K_{xc}$ (for communication from $X$ to $C$).

key is established at both client and node once this setup message is processed.[7] The same alteration to original Tor applies for the extension from $X$ to $Y$ when sent from the client. The result will look something like

$$\{ID_{cx}, EXTEND, DH_{cy,pub}, \{CREATE, ID_{cy}, data\}_{K_{cy}}\}_{K_{cx}} \quad (1)$$

with a reply that uses the new keys $K_{xc}$ and $K_{yc}$[8]. Similar extension is done from $Y$ to $Z$. We have only described the changes to the protocol. There exist checksums and key verification parameters in the current Tor protocol that will fit easily in the same way in the new protocol. Notice that if we use $DH_{cx,pub/priv} = DH_{cy,pub/priv} \neq DH_{cz,pub/priv}$ or $DH_{cx,pub/priv} \neq DH_{cy,pub/priv} = DH_{cz,pub/priv}$ we save one DH initialization and will still have FS as long as the private value is discarded after circuit use. The main reason for not having $Z$'s value equal to $X$'s value is that $X$ and $Z$ should not be able to use the value as an index to trivially tell if they are a part of the same circuit.

This first protocol will serve as the basic building block for those that follow, which are all variants on it.

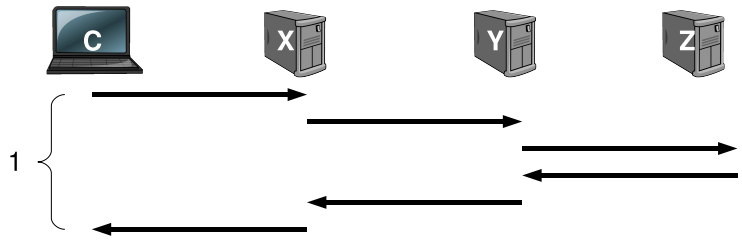


**Fig. 1.** Circuit setup, second protocol.

*The second protocol* creates the complete circuit by sending a single packet to the first server node, $X$, propagating its way through $Y$ to $Z$, as shown in Fig. 1. The initial packet sent from $C$ will look something like

$$DH_{cx,pub}, \{cmd, ID_{cx}, Y, DH_{cy,pub}, \{cmd, ID_{cy}, Z, DH_{cz,pub}, \{cmd, ID_{cz}, data\}_{K_{cz}}\}_{K_{cy}}\}_{K_{cx}} \quad (2)$$

The client (and intermediate nodes) will replace (actually "shift left") the data and add as much padding as they have removed in order to maintain constant data length when stripping off headers. The *data* in Expression 2 can be

[7] Note that since the client contributes only a fresh, ephemeral, and unauthenticated value in this exchange, any concern about Key Compromise Impersonation attacks simply do not apply to our protocols [4].

[8] Recall $K_{yc} \neq K_{cy}$, but they are both derived from equal key exchange material.

parameters to the command for the last node of the circuit, e.g. connect to an external service or a simple "SETUP_COMPLETE" to let the last node know that it should send back a "SETUP_OK". The *data* field will always be followed by termination information and random data adequate to keep a constant length. This is the same technique to hide correlation between onion size and relative position in a circuit used in the two generations of onion routing that preceded Tor [10,20].

This general approach of (1) using just DH in exactly this way and (2) abandoning RSA for circuit establishment was seriously considered [11] by the developers of onion routing when moving from the generation 0 to the generation 1 system design, but ultimately RSA encryption in circuit setup has been used in the code for all three generations of onion routing that have been deployed.

**Preventing replay in circuit setup** If someone were to obtain the onion in Expression 2, he could replay it and cause the exact same circuit to be built with the exact same keys. He could not read any traffic without breaking three private and/or session keys. And, he could only replay it for the lifetime of onion routers' public DH keys. If any of them were to expire and the corresponding private keys be discarded, the nodes would not be able to process the circuit. But during that period, he could build the circuit repeatedly and possibly conduct traffic attacks based on doing so. That is not possible in the current Tor circuit-building protocol because each server contributes an ephemeral DH key to the session keys each time a circuit is built. We now explore how to add this replay preventability.

Following the structure of the first protocol above, we can have the "CREATED" message contain a random value generated by $X$. This can then be combined with the session-key seed to form a new session key. (In current Tor, the key material generated by the DH exponentiations would not be used directly to encrypt messages. Rather a hash of that material with two different known values is used for two different key derivation functions that produce keys for encryption in each direction in the circuit. Similar kdfs are used to produce keys for integrity checks, etc. [6]. As in most publications, our protocol description glosses over this detail.)

This addition to the handshake can similarly be added to the "CREATED" messages from $Y$ and $Z$, resulting in a re-keying of the circuit even as it is being built. Thus, while the original message from the client to $X$ might be replayed, subsequent messages through $X$ will be encrypted under a different variant of $K_{cx}$.

*The third protocol* adds this ephemeral feature to the above protocol designs while reducing the total number of messages and still without requiring any additional exponentiations over the first protocol.

If we were to simply add this node-generated randomness to the second protocol above, it would be possible to rekey the circuit with a single flow up and single flow down the circuit. The full circuit establishment could not be replayed because of this rekeying. But, it would not prevent replay of the onion and of

the resulting path laying all the way through the circuit. An attacker could still replay the onion to do traffic analysis of the circuit establishment as long as the servers' DH keys remained usable.

The current Tor protocol contains a "CREATE_FAST" option for the handshake between the client and the first node. The link between them is already encrypted using TLS (in a DH mode that insures that link encryptions have forward secrecy). Thus, against a link eavesdropper, there is no advantage to using a DH key exchange in the Tor handshake. Therefore both the client and first node simply send each other symmetric key seeds which are combined using XOR to form the Tor session key between them [6].[9]

We can use this technique, but extend it slightly to still reduce the number of ping-pong exchanges used to establish a circuit. The first message from the client is encrypted only with the TLS encryption of the link. It contains an extend instruction and a random value from the client to be combined with a random value contributed by the first node, $X$, to form their session key. Ignoring the TLS encryption, it is very similar to the message in Expression 1 with one extra field and one less layer of encryption.

$$EXTEND, random\_value_{c_x}, DH_{cy,pub}, \{CREATE, ID_{cy}, data\}_{K_{cy}}$$

$X$ forwards this (minus the fields $random\_value_{c_x}$ and $EXTEND$) to $Y$. $Y$ responds with

$$\{ID_{cy}, CREATED, random\_value_{y_c}, data\}_{K_{yc}}$$

to which $X$ attaches a random value and returns to the client

$$ID_{cx}, EXTENDED, random\_value_{x_c}, \{ID_{cy}, CREATED, random\_value_{y_c}, data\}_{K_{yc}}$$

The client then produces $K'_{cx}$ and $K'_{xc}$ from the parameters $random\_value_{c_x}$ and $random\_value_{x_c}$, and the keys $K'_{cy}$ and $K'_{yc}$ are created from $K_{cy}$ and $random\_value_{y_c}$. These keys are used by $C$ as session keys to communicate with $X$ and $Y$ respectively for the remainder of the session. Using the session keys the client sends an "EXTEND" message to $Y$, for extending to $Z$, just as in the first protocol. The entire sequence of exchanges is depicted in Fig. 2.

We achieve a savings of two messages compared to the current circuit construction by creating an onion from the client to $Y$ as shown in Fig. 2, and

[9] One might even go further and question the need for any encryption at all beyond the TLS link encryption for communication between the client and the first node. This would also allow the removal of one ping-pong exchange of handshake messages while otherwise leaving the protocol intact. We will consider just such a reduction next, but without eliminating the exchange of keys. The overhead of keeping these keys is slight, especially if this does not require its own ping-pong of messages, and it provides consistency with other protocol features, hence flexibility. We will thus not pursue further in this paper completely removing the Tor session key between the client and the first node.

then do an extend from $Y$ to $Z$ as above. The initial handshake with $X$ is now bundled in the onion sent to $Y$. And, even if an attacker obtained the onion despite the TLS encryption on the client-$X$ link, he could replay it for at most two hops (and only during the time $DH_{y,pub}$ is valid). He cannot rebuild the entire circuit to the final node because $Y$ will not decrypt the extension request unless it is encrypted under the new key. Note also that this use of a two-hop onion will only allow $X$ to identify its position in the circuit. The circuit will be indistinguishable by $Y$ or $Z$ from one built only by telescoping. This is not a factor in the typical case as most Tor circuits are built from clients not operating on a Tor network node.

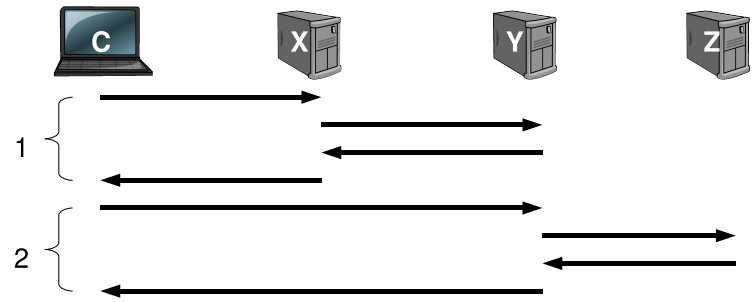


**Fig. 2.** Circuit setup, third protocol.

Another option that will prevent replay while using a single onion to establish the circuit (as in the second protocol), is to use timestamps. If timestamps are added to each layer of the onion, then honest nodes will not process them once they have expired. To be resilient to clock skew, we probably need to have an expiry interval of c. a half hour or hour rather than a few minutes. This however raises the prospect that, e.g., the first node if compromised could replay the onion as much as desired for that hour for whatever traffic-analysis value that could have. Thus, we could have nodes store a checksum of a few bytes or so for all onions that pass through within an hour. It can be quite short given the small likelihood of collision, so neither storage nor lookup should be a problem even for slow nodes without much memory. And, even if our checksum length is so short that each node denies, e.g., ten valid circuits a day because of collisions, the load on the network from circuit setup messages is still greatly reduced, as is the expected setup time for establishing new circuits. Of course timestamps can be added to the third protocol so that even replays just to $Y$ can occur for a shorter period.

Given that most clients are outside the known server node network, it will be trivial for first nodes to recognize themselves as such. In fact, given the use of entry guards in all Tor circuits [17], it is likely that all first nodes can identify themselves with high probability for most circuits. Nonetheless, Tor clients on

Tor network nodes can avoid giving away even this little redundant information by always building circuits using telescoping, even from the first to second nodes. The use of onions (where recommended) saves one exchange of messages by bundling the handshakes of the first and second nodes into one flow up and one flow down the circuit. Note that the use of "CREATE_FAST" to form circuits from a client located on a Tor node using the current Tor protocol for similar reasons faces the same issues.

The *fourth protocol*[10] is useful in cases where forward secrecy is desired not only a week after a circuit is closed but as soon as the circuit is closed. This protocol provides *immediate FS*, whereas the others provide *eventual FS*. In the fourth protocol the number and sequence of message exchanges is the same as in the current Tor circuit establishment protocol (and the same as in the first protocol). The number of exponentiations is much fewer however: eleven vs. eighteen total per circuit, and six vs. nine for exponentiations that cannot be precomputed and must be done during the protocol run. The others can be done in advance during idle cycles. Another virtue of the protocol is that it is compatible with changes that are being contemplated by the Tor developers[11] to improve efficiency in what is stored in directories and how it is distributed. Adding immediate FS to the first protocol in the obvious way of having nodes send back an ephemeral DH public key (as opposed to a random value for modifying the existing session key as in the replay prevention of the third protocol) would not have either of these advantages. If "CREATE_FAST" can be used for the first hop, then the total exponentiations drops from eleven to seven, of which four must be done during the protocol run.

The first message from the client to $X$ is similar to the first message of the first protocol, except that nothing is encrypted (other than by link TLS) because the client is not yet able to form any session key.

$$DH_{cx,pub}, CREATE, ID_{cx},$$

$X$ responds with

$$DH_{xc,pub}, ID_{cx}, \{CREATED, data\}_{K_{xc}}$$

where, if $r_c$ is the client's private ephemeral DH key and $r_x$ is $X$'s private ephemeral DH key, and since

$$(DH_{cx,pub})^{(DH_{x,priv}+r_x)} = g^{r_c \cdot (DH_{x,priv}+r_x)} = g^{DH_{x,priv} \cdot r_c} g^{r_x \cdot r_c} = (DH_{x,pub})^{r_c} \cdot (DH_{xc,pub})^{r_c} = (DH_{x,pub} \cdot DH_{xc,pub})^{r_c}$$

both $C$ and $X$ can use this as key material for the directional keys $K_{cx}$ and $K_{xc}$. The ephemeral key pair $DH_{xc,pub/priv}$ (with $DH_{xc,priv} = r_x$) is formed by $X$ for answering *one* Tor circuit establishment request, and the private component is discarded as soon as it is used to form session keys. Note that the exponentiation

[10] Thanks to Kim Philby for discussions on attempts to break the fourth protocol.
[11] Private communication.

necessary to form the DH key pair does not need to be done during the protocol run. Pairs can be formed and stored during idle cycles of the server. The only exponentiation that must be done by $X$ during the protocol is the one creating key material $(DH_{cx,pub})^{(DH_{x,priv}+r_x)}$. The client also has only one exponentiation to do during the protocol (for each node in the circuit), namely $(DH_{x,pub} \cdot DH_{xc,pub})^{r_c}$ to form the same key material. The basic underlying point is one that is well known to apply to DH protocols in general; nonetheless the current version of Tor does not seem to take advantage of it.

Unlike the first protocol, authentication comes from $X$ being the only one who could encrypt the response rather than being the only one who could decrypt the challenge. In both cases only $X$ possesses $DH_{x,priv}$, and the client knows that $DH_{x,pub}$ is $X$'s midterm DH key from the signed directory information the client has. Thus, the client knows that $X$ is the only one who could form $K_{cx}$ and $K_{xc}$ besides the client, given normal assumptions. (What we have done here is effectively a half-authenticated variant of some existing protocols for authenticated DH key establishment that combine ephemeral and longer term DH parameters, much as our ElGamal key agreement above was a half-authenticated simplification of basic DH; although the exact relation between this protocol and existing ones is not as clear. We will discuss this more in section 5.)

The client next sends the same "EXTEND" message to $X$ for extending to $Y$ as in the first protocol, except that as immediately above, there is no encryption of the message portion arriving at $Y$, other than the link encryption between $X$ and $Y$. $Y$ responds just as $X$ did above. The extension to $Z$ is of the same form as the extension to $Y$.

## 4 Hidden Service Protocol Description

**Using the new circuit protocol in existing hidden service designs** Hidden services can work almost as in the existing deployed design [7,17] only adapting to the new DH-based carrier in the circuits. It is also possible to incorporate so-called *valet nodes* [18], which protect the introduction points from being identified by the client or anyone contacting a directory (or the directories themselves). A server sets up circuit connections to some introduction points and there it listens for connections. Inside the contact information published or given to the client the server adds a valet node extension to the means for reaching introduction points, which is encrypted so that only the valet node knows where the introduction point is. This is completed as described in the just-cited paper [18].

**New Hidden Services setup** One of the major problems with the existing hidden services protocol is that it has become too complex. Both the deployed hidden service design and the design using valet nodes require the building of four circuits collectively comprised of as many as twelve Tor server nodes—not including the service lookup, the client, or the hidden service node itself. Extending design ideas from the proposed addition of valet nodes to protect

the introduction point [18], we here propose how to drastically reduce both complexity and latency when connecting to a hidden service.

Connection between a client and a hidden service requires the setup of two separate paths, each comprised of two mated Tor circuits. Why incur the large overhead cost and delay of the second pair of circuits and connection made through the rendezvous point? Rendezvous circuits provide at least three things in the deployed hidden service design. (1) Introduction points are not responsible for serving up the contents of the hidden server for which they are introduction points. (2) Hidden servers (and thus the network) do not have to maintain open circuits adequate to carry the maximum number of simultaneous connections they might have. (3) None of the nodes carrying traffic between the client and hidden server can recognize that they are carrying traffic for that hidden service. In particular, blocking of an introduction point is neither as significant nor therefore as desirable for an adversary wishing to deny service provided by the hidden server.

The introduction of the valet nodes and contact information changed this, as we now can have valet nodes protecting the introduction points. And contact information is structured and served such that it requires some effort for either valet nodes or introduction points to determine any hidden service for which they are valet or introduction nodes respectively, even if the hidden service is publicly listed. Since there are several of each, the value of determining this is also limited and thus less likely to be pursued. We propose herein to further change the introduction points into contact points where the service can either (1) remain connected to the client rather than opening a new connection (Fig. 3), or (2) set up a new connection to the node preceding the valet node, using this as a rendezvous point (Fig. 4).

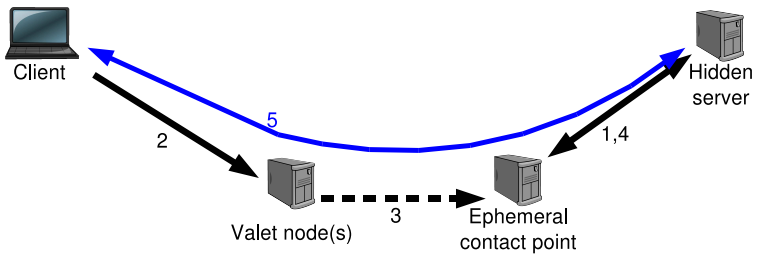


**Fig. 3.** New direct hidden service usage.

The first scenario from Fig. 3 follows simple setup. First the service opens up connections to contact points (1) and tells them to listen for connections. Then it locates the valet nodes and produces the contact information which is

somehow[12] received by the client. The client tunnels out to the valet node and transfers the valet ticket (2). The valet node unpacks the valet ticket, and extends the tunnel to the ephemeral contact point (3). After using the valet ticket information to authorize himself to the contact point, the contact point submits (4) client information to the hidden service and connects the two circuits (5) for the client and the hidden service talking directly. This is much faster but has some implications that we will discuss in section 5.

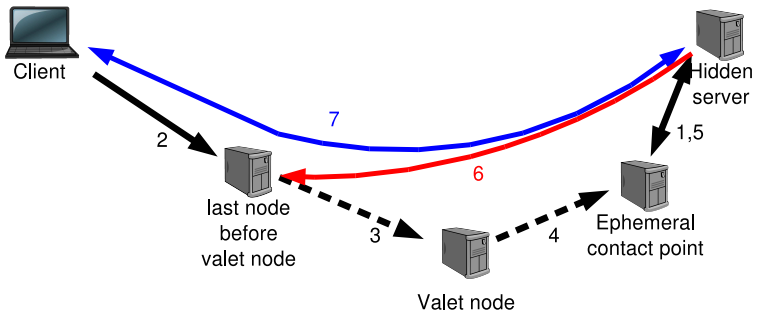


**Fig. 4.** New direct hidden service using new rendezvous point.

The second scenario shown in Fig. 4 is optional and could e.g. be used when connecting to a public hidden service. The first part of the service setup is the same, but the client now first constructs a tunnel to the last node in front of the valet node (2) and asks this node to listen for a potential connection request. This node will now act as a potential rendezvous point for connections back from the hidden service. Then the client extends to the valet node (3), informs the contact point (4) which authorizes and submits the information to the hidden service (5). The hidden service now determines whether to establish the connection through the valet node as in the first scenario, or to contact the new rendezvous point (6), and the client and the hidden service will have their new communication channel (7).

Note that, as described, the two scenarios need not be considered two entirely distinct protocols, but rather dynamic options. As observed earlier, the first scenario addresses all of the contributions of using rendezvous points except possibly for the overhead caused by the number of open circuits that a hidden server must maintain to remain reachable. The network overhead can potentially be reduced somewhat already by the suggested approach since circuits to the contact points may be shorter because the valet node is also chosen by the hidden server. As just observed the hidden service can dynamically choose whether to communicate with the client through the already opened circuits or to open a

[12] Could be off-line distribution, or via a directory service, DHT, etc.

new circuit to the offered rendezvous point. If the server has adequate reserve contact circuit sockets and bandwidth, it can use the open circuits. If not, it can use a new circuit to the rendezvous point, thus addressing the open circuits issue, but doing so in a more efficient, dynamic way.

## 5 Discussion

### 5.1 Calculation reduction

We are ignoring the TLS encryption overhead as this is expected to be almost like persistent connections and these will exist in all protocols in any case. We will also ignore the symmetric encryption and the signature verifications as they are the same in the two versions. The servers will have the additional production of public DH values for authentication during every rotation period, which safely can be ignored: we assume a default rotation of one per week.

*The current version of Tor* uses an RSA encrypted DH key exchange including generation of DH public and private values when setting up a new circuit to another node. The client uses an RSA encrypted DH key exchange including generation of DH public and private values. So the client makes an RSA-encryption only once and adds a symmetric key if the DH key material is too long for one RSA encryption (which it currently is). On the server node's part there is the decryption of the RSA data and add-on key, decryption of the DH key material, and calculation of the keys[13]. Some of these calculations can be made on idle circuits as noted in Table 1, but to our knowledge the current Tor protocol does not take advantage of this: all calculations are done during the circuit establishment protocol. The "CREATE_FAST" option in the current version will result in one less DH key exchange and one less RSA encryption/decryption.

*Our new proposal The second protocol* uses two DH value generations on the client side (one for $X$ and $Y$, and one for $Z$) but these can be made on idle cycles, and since the server nodes' DH values are known through the data from a directory service, the client can complete the DH-key generation and calculate the keys from this value. The client then encrypts the rest of the data with the correct keys immediately. The client avoids the need for RSA encryption for each node. In addition the nodes do not need to decrypt the RSA data. They generates the keys directly. In addition the nodes need not send the initial DH-value since the client uses its public DH values, as shown in Table 1. *The fourth protocol* also has two initial exponentiations for the client, but for these and for each of the circuit nodes the initial exponentiation can be done on idle cycles, and these should therefore not count in comparing resources. The client will also have to finalize all three temporary session keys by doing one exponentiation each, and so will each of the circuit nodes. The "CREATE_FAST" option to the fourth protocol will result in one less DH key exchange. Unlike for

[13] The DH-key is used to generate multiple keys for both encryption and MACs.

the second protocol, the client should use the same ephemeral public DH key for $Y$ and $Z$, to save an exponentiation from this reduction. So there will be one initialization at the client, and none at $X$. As with the current Tor circuit protocol, "CREATE_FAST" should not be used if the circuit is initiated by a client at a directory-listed Tor node.

| Calculation type in a three node circuit | Current design | Current FAST | Second protocol | Fourth protocol | Fourth FAST |
|---|---|---|---|---|---|
| # of client RSA encryptions | 3* | 2* | 0 | 0 | 0 |
| # of nodes' RSA decryptions | 3 | 2 | 0 | 0 | 0 |
| # of client DH-initializations | 3* | 2* | 2* | 2* | 1* |
| # of nodes' DH-initializations | 3* | 2* | 0 | 3* | 2* |
| # of client DH-finalizations | 3 | 2 | 3 | 3 | 2 |
| # of nodes' DH-finalizations | 3 | 2 | 3 | 3 | 2 |

**Table 1.** Number of exponentiations calculated during a single circuit setup. **These initializations can be preconstructed on idle cycles.*

### 5.2 Location Hidden Service effects

**Using new DH circuits on the currently deployed hidden service design** As most circuits used by a client are premade to at least two hops out, there would be no noticeable change to the user experience from the existing hidden service design. But every circuit initialization will save the network the number of exponentiations reflected in Table 1. And as we still have three new circuits opened for every connection to a hidden service our new protocol reduces the number of exponentiations in the network significantly.

**Using new circuits and valet nodes** If the new circuit setup were implemented on the valet nodes design an estimated reduction in calculations would be the same as in the existing functionality. Even if our protocol suggestion supports and makes implementation of valet nodes easier, we will only see the same amount of latency as in the original version. The only difference is that the valet node can be based on a half-finished DH exchange and therefore may also replace the RSA encryption of the valet token.

**Using new circuits and direct communication** One of the primary objections to using the old introduction points as contact points for the hidden service, was that they might become liable for the content of the hidden service. The introduction of the valet nodes changed this because the introduction points no longer know which service they are assisting. But now the valet nodes could identify themselves as associated with a service, *if* they had access to the contact information for the hidden service. In addition, there could be many

valet nodes per introduction point, so we estimate that the potential problem of being blocked by some valet node is not likely to be critical for the hidden service. When we are talking about really hidden services that have private contact information this is no issue at all.

By dynamically choosing whether to communicate through the contact and valet circuits or open a new circuit to the rendezvous point (node before the valet node in the valet circuit), the hidden server can more effectively manage the network costs of connections to hidden services. Note that the incentives of the hidden service align with those of the network in that it is incented to only open new rendezvous circuits when utilization of its contact circuits is relatively high. It would be interesting to investigate further whether the optimal choice of resources in terms of number of open circuits to contact points maintained vs. percentage of rendezvous circuits needed is the same for a given hidden server and the network it is on. Clearly different principals in the system also learn different things about the relative load on a hidden service from the dynamic choice of whether to create a rendezvous circuit (e.g., the valet, contact potential-rendezvous, and guard nodes). Whether there is any significant information discernible from that (and whether it would be discernible in the currently deployed hidden service design) is another interesting question worthy of further study.

### 5.3 Security

**Forward Secrecy and Replay** Perhaps the largest security change from current Tor implied by all the protocols except the fourth is that the FS they offer is eventual rather than immediate. If DH keys for server nodes are used for a week and kept for two, as would be consistent with existing directory usage in Tor, then it can be as much as two weeks from the time a circuit is initiated until the session keys in it attain FS; although, it will typically be much shorter. This has two effects: first is the replayability of circuit setup for traffic analysis purposes and the vulnerability of circuits to an adversary that attacks nodes along a circuit during the lifetime of the DH keys to uncover traffic and data, up to potentially everything sent over a circuit. Only a protocol with eventual FS is vulnerable to replay once the circuit closes. However, as we showed via the third protocol, it is possible for an eventual-FS protocol to be vulnerable to an attack on servers or keys before the FS takes effect but still not be vulnerable to replay. An adversary willing to go to the effort of such traffic analysis as can be obtained from replay probably is determined enough to attack servers and keys as well. This is the reason that we recommend the fourth protocol as a new Tor circuit protocol rather than the third even if the third is resistant to replay. Nonetheless, for the vast majority of Tor traffic, both of these concerns are beyond a reasonable threat model.

**Authentication and Protocol Security** The current Tor circuit protocol was designed to fit message constraints that "a single cell is too small to fit

both a public key and a signature" [7]. It was thus forced to use a nonstandard design. For this reason, it was analyzed by the NRL protocol analyzer before it was deployed and found to be secure in the Dolev-Yao model [7]. In 2005, the Tor developers noticed and corrected that the cryptographic instantiation of the protocol failed to properly perform adequate checks and left circuit building subject to significant attacks. Analysis by Ian Goldberg [9] showed that the corrected protocol instantiation was secure in the random oracle model. What assurances do we have that the protocols we have presented are secure? At this point we have only indications, which we now discuss.

Using DH, we do not have the message size issues of the current Tor circuit protocol, but we have as yet performed neither formal analysis nor a cryptographic proof of the security for any of our protocols. Nonetheless, all but the fourth protocol are essentially ElGamal key exchange. This is a widely studied and understood simple protocol for providing implicit one-sided authentication. As such, these protocols are unlikely to have significant flaws. The fourth protocol combines long-term and ephemeral DH elements in a manner similar to many protocols, but again for only one-sided authentication so that it is simpler. It is in some ways like a simplification of the MQV protocol [14]. It has the overt structure of MQV to authenticate the server node to the client and obviously none of the structure authenticating the client to the server. Our protocol also does not make use of the specialized group exponentiation that MQV uses. Like ElGamal, MQV is also a well-studied protocol. Its original design was vulnerable to attacks against properties that are not needed for our purposes and were later corrected and led to its adoption as an IEEE standard[12]. Despite adoption as a standard, MQV has not been proven secure either formally or cryptographically. The only protocol in this group to have a security proof is the so-called Unified Model (UM) protocol [4]. Adapting the UM protocol in a straightforward way to our purposes would increase the number of exponentiations required vs. our fourth protocol. While similarities to UM and other protocols is encouraging, we intend to subject our protocols to more formal scrutiny in future work.

## 6 Conclusion

We have proposed a way to simplify circuit setup in the Tor anonymizing network. We have explained how to use predistributed Diffie-Hellman values for setting up session keys based on half-certified ElGamal key exchange. By using this new setup for a circuit the client saves three RSA encryptions, and each of the nodes in the circuit saves one RSA decryption in addition to the initialization of a DH value. In addition we noted how both the current Tor circuit building protocol and our new proposed protocols can benefit from precomputation of much of the information needed for the protocols. This is perhaps especially beneficial at the nodes rather than clients, where public-key overhead can be a bottleneck. One of our protocols offers less calculation overhead, and incorporates immediate forward secrecy. Others provided even more substantial savings in computation and in communication but only eventual FS. They also serve

to illustrate the distinctions between eventual FS, replay-resistant eventual FS, and immediate FS.

We have also proposed two new hidden service protocols that uses valet nodes to protect the introduction point, and therefore can eliminate the circuits to external rendezvous points. As a result of this improvement the hidden service protocol can now make more direct, lower-overhead connections to hidden services without compromising on anonymity or security.

## References


1. The Anonymizer. http://www.anonymizer.com/.
2. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.
3. Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.
4. Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.
5. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.
6. Roger Dingledine and Nick Mathewson. Tor protocol specification. http://tor.eff.org/svn/trunk/doc/spec/tor-spec.txt, February 2007.
7. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
8. Tahir ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*, 31(4):469–472, July 1985.
9. Ian Goldberg. On the security of the Tor authentication protocol. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, UK, June 2006. Springer.
10. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.
11. Onion Routing: Brief Selected History. http://www.onion-router.net/history.html.
12. IEEE. P1363 standard specifications for public-key cryptography. IEEE Std 1363-2000, January 2000.
13. Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based onion routing. In *Proceedings of the Seventth Workshop on Privacy Enhancing Technologies (PET 2007). (This proceedings.)*. Springer-Verlag, LNCS, 2007. Also University of Waterloo, Tech. Report CACR 2007-08.
14. Alfred J. Manezes, Minqhua Qu, and Scott A. Vanstone. Some new key agreement protocols providing implicit authentication. In *Workshop in Selected Areas of Cryptography (SAC'95)*, pages 22–32, 1995.

15. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
16. Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, pages 27–36. ACM Press, November 2006.
17. Lasse Øverlier and Paul Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.
18. Lasse Øverlier and Paul Syverson. Valet services: Improving hidden servers with a personal touch. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, UK, June 2006. Springer.
19. Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for Anonymous Routing. In *Proceedings of the 12th Annual Computer Security Applications Conference*, pages 95–104. IEEE CS Press, December 1996.
20. Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.
21. Relakks. http://www.relakks.com/.