Apres - A System For Anonymous Presence

Ben Laurie

Step 2, 17 Perryn Road, London W3 7LR, United Kingdom ben@algroup.co.uk

Abstract. If Alice wants to know when Bob is online, and they don't want anyone else to know their interest in each other, what do they do? Once they know they are both online, they would like to be able to exchange messages, send files, make phone calls to each other, and so forth, all without anyone except them knowing they are doing this.
Apres is a system that attempts to make this possible.

1 Assumptions

I assume a messaging system that supports anonymity. I assume that there is an untrusted (but cooperative) server that both Alice and Bob can access. I assume that Alice and Bob have communicated by some other channel prior to using this protocol[1].

2 Aims

Even if Alice frequently announces her presence to Bob, then neither the server nor an observer should be able to either link any such announcements with each other, nor know when Alice is logged on, nor know that Alice and Bob communicate.

If Alice announces her presence to Bob and Charlie, then Charlie and Bob should not be able to demonstrate to each other that they are both friends of Alice[2].

3 Definitions

We construct an implicit address (as first suggested by Karger[4] p.111 and forward) for each pair of communicating entities, as folows. The implicit address changes with time in order to confound traffic analysis.

If two entities A and B share a secret (known as an *ID*, since it is used to identify A to B and vice versa), $ID_{A+B} = ID_{B+A}$, then for any particular time period T there is an *ID de jour*:

[1] But note that I do not assume they managed to exchange a strong secret beforehand.

[2] Of course, they can reveal that they have both have friends who are in communication with the server at the same times, assuming Alice chooses to always announce her presence accurately to all her friends

$$IDJ_{A+B,T} = H(T, ID_{A+B}) \tag{1}$$

Where $H(x, y)$ is an HMAC of x with key y. This construction is used to make the shared ID unlinkable between time periods.

A and B have to somehow generate this shared secret, and may not be in an environment that permits the secure exchange of large random numbers. So, they use an *introduction ID*, $Intro_{A+B} = Intro_{B+A}$. This is typically a relatively weak secret whose only purpose is to allow A and B find each other amongst other potential partners. Like IDs, an introduction ID also can be used to construct an *introduction ID de jour* for time period T.

$$IntroDJ_{A+B,T} = H(T, Intro_{A+B}) \tag{2}$$

A modified Diffie-Hellman key exchange[1] is used: A sends B $g^{r_A} \pmod p$, B responds with $g^{r_B} \pmod p$, then the secret each calculates is

$$g^{r_A r_B H(Intro_{A+B})} \pmod p \tag{3}$$

The purpose of this is to make the exchange resistant to man-in-the-middle attacks (since the man-in-the-middle doesn't know $Intro_{A+B}$).

Whenever I use the word "sends" I mean "uses a mechanism which protects the anonymity of the client to transmit a message", such as Tor[2], Mixminion[3] to a newsgroup or Freenet.

4 Introduction Protocol

Unlike the presence protocol which will be described later, the introduction protocol is not expected to require both parties to be online simultaneously. Each segment of the protocol is executed asynchronously, and the server stores information as required. What stage in the protocol A, B and the server, S, have reached can be deduced entirely from the data currently stored on the server and the two clients, as can the roles of A and B (that is, if B connects first, then effectively B becomes A and A becomes B). The protocol is also designed to handle loss of state by any or all of the parties. Each message consists of a message type, M_x and data which depends on the message type.

As part of the protocol is a Diffie-Hellman key exchange, there is a well-known generator, g, and prime, p.

In every case, the client initially makes a connection to the server, and sends at least an introduction ID de jour and half of a Diffie-Hellman key exchange. For the sake of clarity, I will assume, without loss of generality, that A makes the first connection, and sends:

$$A \rightarrow S : M_{I1}, IntroDJ_{A+B,T}, g^{r_A} \pmod p \tag{4}$$

Where r_A is a random number chosen, stored and reused in every subsequent connection by A. Since this is the first connection, the server knows it because

it has no information associated with $IntroDJ_{A+B,T}$. The server now stores $g^{r_A} \pmod p$. If A reconnects, then this is detected because $g^{r_A} \pmod p$ is the same as is already stored and nothing else is stored (i.e. no input from B); in this case, the server responds with an empty message, M_{IR4}.

At some later point, B connects for the first time, and sends:

$$S \leftarrow B : M_{I1}, IntroDJ_{A+B,T}, g^{r_B} \pmod p \tag{5}$$

The server knows that this is the second connection, because $g^{r_A} \pmod p$ is already associated with $IntroDJ_{A+B,T}$ and is different from $g^{r_B} \pmod p$. Note that if this is, in fact, A reconnecting having lost r_A and generated a new random number, then the protocol will eventually recover so long as the server always stores only the two most recent different $g^{r_x} \pmod p$ it has seen. In response to this message, the server sends $g^{r_A} \pmod p$:

$$S \rightarrow B : M_{IR1}, g^{r_A} \pmod p \tag{6}$$

B can then calculate the shared secret

$$S_{A+B} = g^{r_A r_B H(Intro_{A+B})} \pmod p \tag{7}$$

Using this secret, B can derive a symmetric encryption key K_{A+B}, for example using SHA-256 of $S_{A+B}||'C'$[3]. B then sends proof that it knows $Intro_{A+B}$ and a textual message, $T_{B \rightarrow A}$.

$$S \leftarrow B : M_{I2}, H(T, Intro_{A+B}||'A'), E_{K_{A+B}}(T_{B \rightarrow A}) \tag{8}$$

The server cannot, of course, verify the proof, so it merely stores it, ready for the next time A connects. At this point, A sends M_{I1} as above, and the server responds:

$$A \leftarrow S : M_{IR2}, g^{r_B} \pmod p, H(T, Intro_{A+B}||'A'), E_{K_{A+B}}(T_{B \rightarrow A}) \tag{9}$$

It might appear that knowing K_{A+B} is sufficient to prove knowledge of the ID, but then A or S could replay B's message without knowing the ID. At this point, A checks $H(T, ID||'A')$, and, if it matches, replies with its own message (possibly after interaction with the user, though that could wait until the protocol is complete):

$$A \rightarrow S : M_{I3}, H(T, Intro_{A+B}||'B'), E_{K_{A+B}}(T_{A \rightarrow B}) \tag{10}$$

Once more, the server stores $H(T, ID||'B')$ and $E_{K_{A+B}}(T_{A \rightarrow B})$. Next time B connects, sending M_{I1} again, the server sends:

$$S \rightarrow B : M_{IR3}, g^{r_A} \pmod p, H(T, Intro_{A+B}||'B'), E_{K_{A+B}}(T_{A \rightarrow B}) \tag{11}$$

[3] 'C' is chosen because 'A' and 'B' are used elsewhere in the protocol.

B checks $H(T, ID||'B')$, and if it matches, the protocol is complete, A and B have a shared secret, S_{A+B}, and it the users are happy with the textual messages exchanged, they can proceed to use the presence protocol.

5 Presence and Instant Messaging Protocol

Once A and B have established their strong shared secret, it is assumed this is good for all time[4]. The first problem they face, though, is that they want to communicate without compromising their anonymity. One of the many possible ways to do this is with instant messaging. This requires a server – though all the server really does is provide filtering, so communications are $O(n)$ instead of $O(n^2)$ (where n is the number of users)[5]. Once more, the ID de jour is used.

Assume A has a set of friends, $\mathcal{F}_A = (A_0, A_1, \cdots, A_N)$. Then A constructs the list of IDs de jour for the friends, $IDJ_{A+\mathcal{F},T} = (IDJ_{A+A_0,T}, IDJ_{A+A_1,T}, \cdots, IDJ_{A+A_N,T})$ (where $ID_{x+y} = S_{x+y}$) and sends it to the server

$$A \rightarrow S : M_{P1}, IDJ_{A+\mathcal{F},T} \tag{12}$$

the server keeps the list and each time someone else connects, compares their list to all the stored lists. For each ID de jour that matches one on an existing list, say, $IDJ_{A+A_n,T}$, it sends that ID to each of the connections with the matching ID (including the new connection)

$$A \leftarrow S : M_{P2}, IDJ_{A+A_n,T} \tag{13}$$

$$A_n \leftarrow S : M_{P2}, IDJ_{A+A_n,T} \tag{14}$$

note that there should normally be only be two connections with the same ID[6]. At this point, A and A_n know that each other is online.

At any time, A can send a message to the server to be relayed to any one of $\mathcal{F}_A$ (or, indeed, any other ID de jour)

$$A \rightarrow S : M_{P3}, IDJ_{A+A_k,T}, X \tag{15}$$

where X is anything. The server immediately relays it to all other connections with the id $IDJ_{A+A_k,T}$

$$A_n \leftarrow S : M_{P4}, IDJ_{A+A_k,T}, X \tag{16}$$

X can convey an instant message, information allowing a direct connection, or anything else the two parties care to exchange. X will normally be encrypted

[4] Of course they are free to throw it away and choose a new one at any time.

[5] Note that if we can accept $O(n^2)$ behaviour, then the IDs can be made one-way – that is, such that A can tell B is online without revealing that A is online. Also, we can use non-specialised messaging servers, such as IRC servers.

[6] If there are more, something has gone seriously wrong, or the same person is connected more than once

$$X = E_{K_{A+A_k}}(X') \tag{17}$$

if A_k is not online, then the server could store the message, but note that it will become useless at the end of the time period – although A_k could reconstruct the appropriate ID in order to retrieve the message, doing so in the obvious way would make A_k's sessions linkable, so it would require mechanisms I will describe elsewhere.

5.1 Connection Discipline and Cover Traffic

Although Tor provides anonymity for the client (and could also provide it for the server, but we don't require that), the client *must* use a separate anonymised connection for each instance of any of these protocols. Furthermore, separate connections should also be used for each different time period. If this is not done, then the IDs de jour for different time periods will be linkable, either by the server or by an observer of of the server's connections, by virtue of being used in the same connection.

Because switching from one connection to the next at the end of the time period would also allow the connections to be linked, the client should start each new connection at some random time before its time period actually starts. In order to reduce the number of connections to the server I suggest the random time should be in the range $[T/10, T/2]$ where T is the length of the time period. Because introductions are only used when first establishing a connection between two people, there is little point in disguising the switchover between time periods.

While the user has two connections operating, they should both be used, otherwise the transition from one to the other can be used to link the sessions. The one that is not actually in use should have cover traffic sent to it at random intervals, where the intervals have the same profile as the real traffic (alternatively, messages could be sent down random choice between the two channels, but cover traffic would still be needed to hide the transition). Note that "in use" means when the person (or people) at the other end of the connection have also started using the ID for the next time period, and the cover traffic should correspond to the volume appropriate for those connected.

There's no particular point in keeping an old connection open, unless one wants to allow for clock skew.

Depending on the application, cover traffic might also be desired during normal use, since otherwise the two links could be connected by having the same traffic profile – without evidence, I suspect that instant messaging has too variable a profile for this to work, but this is an area for further study.

5.2 Wire Format

In my current experimental implementation of Apres, messages are sent with fields space separated, and all fields except the message type encoded in hexadecimal. Each message is terminated by a newline.

Future versions will almost certainly use a binary format to reduce message size.

5.3 Introduction ID

If the two users each have access to a computer when they decide to make their introduction, then they can use some kind of strong randomness for their shared secret – but they must ensure that this is not overheard. If they already have some secure channel, such as PGP or encrypted instant messaging, then they can easily do this, otherwise they must establish a secure channel.

A more awkward case is a face-to-face meeting where one or both parties do not have a computer. Writing down a large secret (or even generating one) is annoying and error-prone. One protocol for these situations[5] would be for each person to choose two words. Both people then remember (or write down) all four words. Assuming people make some effort to choose from a wide vocabulary, we could safely assume around 12 bits of entropy in each word, giving a total entropy of 48 bits. In order to avoid confusion, the words should probably be used in alphabetical order, so there is no entropy in their ordering.

5.4 Choice of Time Periods

The time period for introductions needs to be sufficiently long that two users can actually reliably exchange messages without necessarily being online at the same times, but short enough that their shared secret is unlikely to be duplicated, and also short enough to prevent a brute-force attack by a man-in-the-middle on their introduction ID. If we assume that the weak IDs suggested above are used, then this gives us an upper bound for this time period. Assume the attacker can try N IDs per second, then the total time should be much less than $2^{48}/N$ seconds. Assuming an extremely capable attacker that can attempt 1 million IDs a second, this is around 3,000 days.

Duplication is a little more restrictive – if there are 2^{24} IDs in use, then the chances of two colliding are approximately 1 in 2. If there are M new IDs per day, then we want the period to be less than $2^{24}/M$. Making the optimistic assumption of 1 million new users per day, then this gives 16 days.

So, a time period of a few days to a week would seem quite safe for introductions.

As for messaging time periods, these should be changed as often as possible - the only real consideration being the load on the server caused by the new connections required.

6 Security Considerations

Since $Intro_{A+B}$ is not a strong secret, an attacker could guess it, by brute-forcing $IntroDJ_{A+B,T}$. Because the protocol uses a modified Diffie-Hellman key exchange to give a shared secret, the attacker needs to attack that by using a

man-in-the-middle attack. However, he cannot complete this attack (because of the use of $H(Intro_{A+B})$ in the production of the final secret) until he knows $Intro_{A+B}$. Until then, he would be unable to produce the encrypted messages or hashes of $Intro_{A+B}$ used in the final stages of the protocol. So, A and B must complete their initial exchange before the attacker has completed the brute force attack. Brute forcing the introduction ID after the fact does not help, because A and B have by then established a strong secret, S_{A+B} which cannot be derived from the messages exchanged.

Another consequence of the low entropy of this ID is that it is very important that A and B include identifying information in the messages $T_{A\rightarrow B}$ and $T_{B\rightarrow A}$ so they can verify that they are, indeed, talking to the correct person. These messages are, of course, readable by anyone who happens to have chosen the same introduction ID, so the content should take into account the unlikely eventuality that a third party might read it (of course, they are unlikely to respond such that the connection will be used, so at worst they should only know the identity of one of A and B).

7 Future Directions

A robust user-friendly implementation of the protocol (I already have experimental code working).

Implementation of serverless versions of the protocol (using, for example, NNTP as the transport, or distributed hash tables).

8 Acknowledgements

My thanks to Matthias Bauer for useful feedback on early drafts.

References

1. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
2. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router, 2004.
3. David Hopwood, George Danezis, Nick Mathewson, and Roger Dingledine. Mixminion: Design of a type III anonymous remailer protocol, July 11 2002.
4. P. A. Karger. NON-DISCRETIONARY ACCESS CONTROL FOR DECENTRALIZED COMPUTING SYSTEMS. Technical Report MIT/LCS/TR-179, Massachusetts Institute of Technology, May 1977.
5. Cliff Skolnick. Personal communication.