

SAMPLED

Shared Anonymous Music PLayer using wireLess Devices

Dartmouth Computer Science Technical Report TR2006-577

Constantinos Neophytou
Advisor: Tristan Henderson

June 7, 2006

Abstract

Recent advances in mobile computing enable many new applications, yet at the same time create privacy implications caused by the increasing amount of data that becomes available. This thesis will explore the possibilities of wireless-enabled portable devices and their attending privacy implications. We will describe how such a device containing personal information about the musical preferences of its user can help improve the user's experience in a social setting where music is played for all, and at the same time preserve each user's privacy.

1 Introduction

Social networking – connecting individuals or organizations through various social familiarities using a variety of tools[14] – has been at the forefront of recent advances in Internet applications. Community sites, such as MySpace.com, and bookmark sharing sites, such as del.icio.us, help bring people together, but also serve as a source of new and interesting information.

Recommending products and services based on data supplied by users with similar tastes and interests is not a new idea. Amazon.com has been recommending products to its customers based on similar purchases and searches that other users have performed for years. Last.fm, with their Audioscrobbler application, draws upon this same idea. Having the ability to track listening habits by monitoring the titles/artists of the songs users listen to, Last.fm is in a position to use the information they collect from all users and to determine what other songs each user might be interested in listening to. By building a large database of users' listening habits, Last.fm can provide more accurate suggestions and detect new trends.

In addition to the development of these software technologies, there is increasing interest in mobile devices. From Blackberries, to PDAs, to large-capacity MP3 players (such as iPods), people are carrying their data (such as music) with them more often than ever before. As a result, we start seeing applications like Push!Music, researched and developed by a Swedish group, described as a “mobile, peer-to-peer music listening and sharing application that is supposed to act as a source of inspiration for listening to new music and as a new way of recommending music to other people”[8]. The application, which runs on Wi-Fi enabled PDAs, automatically sends recommended songs to nearby users. Push!Music can also receive spontaneous music recommendations from those users.

Assuming that this increase in the number of people who use mobile music players will continue, and that technological advances will eventually open up the mass market to wireless-enabled portable devices that store a large amount of data about their owner's music preferences (for example, a wireless enabled

iPod), it is not unreasonable to believe that applications that take advantage of these features to enhance the user's experience will start to emerge.

For instance, suppose Alice is the owner of an establishment (say a club, a bar, or a coffee shop) that serves patrons. This bar is a location where people congregate, talk, and relax. Alice would likely have music playing in the background. This music may be necessary to fill up moments of silence during conversations (like in coffee shops), or as a main attraction (such as a rock music-only bar). We assume that if the patrons of the establishment enjoy the music they hear there, they are more likely to enjoy the time they spend in Alice's bar, which will in turn translate to higher revenues for Alice.

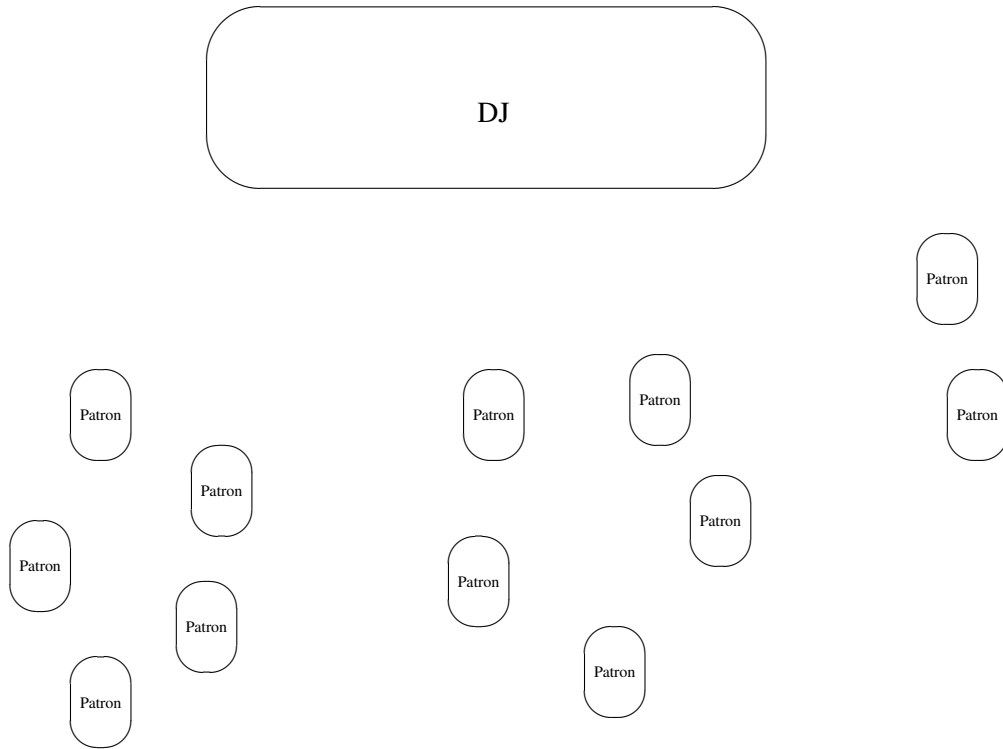


Figure 1: Birds-eye view of Alice's bar showing the DJ and several patrons

If we assume that Alice's patrons all own some wireless-enabled device that either contains their music, or simply their musical preferences, then it is not far fetched to believe that a centralized system can exist that will read those preferences and adapt the bar's playlist to match the patrons' music styles.

However, several problems, arise from this assumption, most of them related to privacy. Some patrons may not feel comfortable with a server they have no control over accessing their listening habits. With the increasing popularity of Internet and network applications, buzzwords such as 'anonymity,' 'confidence,' 'trust' and 'privacy' are on the minds of more and more people. Real world examples of user response to privacy concerns can be found in the section 5.

In addition to privacy concerns, we have security concerns. Alice is the owner of the bar, but the playlist of that establishment is effectively controlled by an automated system that receives recommendations from small, remote units. What would prevent Alice's rival bar owner across the street from flooding her rock-only playlist with opera and country songs, effectively driving all of Alice's rock music listening clientele away?

This paper describes a system that enables Shared Anonymous Music Playback using wirelEss

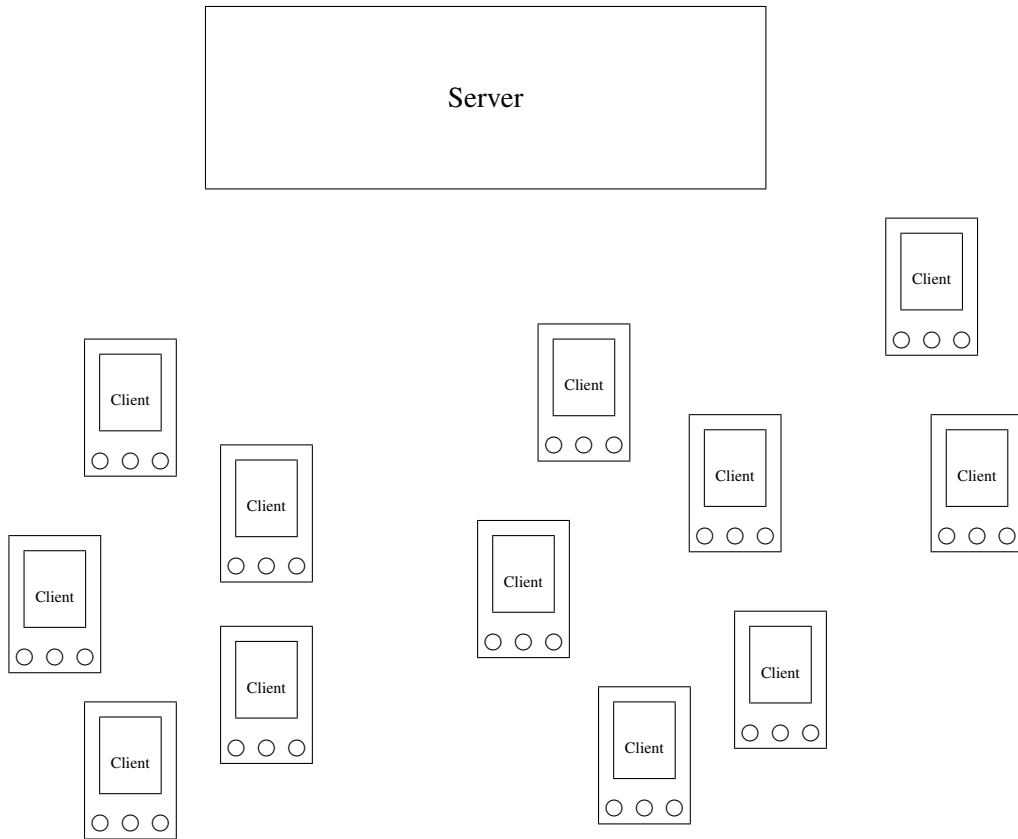


Figure 2: Birds-eye view of Alice’s bar showing centralized server and wireless-enabled devices (Clients) instead of patrons

Devices (SAMPLED), which addresses all the privacy and security issues that are inherent in its design using existing technologies that are freely available.

2 Requirements

2.1 Usability

SAMPLED depends on the information the patrons provide through their devices. Assuming that their favorite songs (or, at least, songs they like) are located on those devices, then the dependency is satisfied completely. Alternatively, each patron (or user) is able to manually feed a list of favorite or preferred songs into their device. The key for a successful implementation, however, is minimal interaction. The patron cannot be expected to interact with the device while in the bar for the system to operate, since that would reduce the quality of the patron’s bar experience.

2.2 Privacy and Attack resistance

As suggested in the introduction, such a system raises concerns about the privacy of its users, but also about the security and integrity of the system itself. The areas that need to be safeguarded fall into several categories.

2.2.1 Trust Model Summary

Given the nature of the setting where such a system would operate, the trust model for the architecture is very limited. The users own the device that holds their music, and we can assume that this device is secure (we will note later what happens if it is not). For SAMPLED to work, users will need to install and run some software on their devices that will interact with the server. Therefore, the users need to trust that the required software will never violate their privacy by sending their personal data to the server. They also trust that this action will cause the server to modify the playlist of the bar in such a way that will include songs those users will enjoy more. However, as far as each user is concerned, nothing beyond their device is trusted, including the wireless network, the other devices that are operating in the system, and even the server itself.

From the server's point of view, a similar model applies. The devices are outside of the server's control, and similarly the wireless network can also not be trusted. Therefore, any measures that are taken to ensure the user's trust of the server is not violated need to take this trust model into consideration.

2.2.2 Threat Model Summary

From our limited trust model, we can assume that only one of the end systems is secure at any given point, that end system being the one over which we have complete control (the server or a user device, depending on our point of view). Additionally, we assume that any adversary has more or less complete control over the communications channel.

The following sections categorize all the possible threats based on these models.

2.2.3 Privacy and passive attacks

1. Anonymity

When a patron is connected to and using SAMPLED, it should not be possible for anyone to retrieve any personal information that can be directly connected to that patron. The possible retrieval methods might include sniffing the wireless network, as well as accessing the information that is processed by, and stored on, the server.

2. Linkability

The development of SAMPLED will inherently be time and location aware. This means that careful observation and tracking of the patrons entering and leaving bars that use SAMPLED can assist an adversary to break the anonymity by tracing similar anonymous data packets that were observed at different times, and correlating those observations with the physical traffic of the bar at that time.

Furthermore, that same traffic can be correlated not only over different times, but also over different locations. By observing the data being sent at different bars that use SAMPLED, it might be possible for an adversary to track the movements of a single patron through various nights, which would violate that patron's privacy. Therefore, the system needs to protect against linkability.

3. Repudiability (Non-repudiation)

Non-repudiation refers to the ability of the recipient of a communication to conclusively verify the identity of the sender. The implication might be that SAMPLED would need to disallow non-repudiation to ensure privacy. As we will explain, however, repudiability is irrelevant to our application. Only the personal data supplied by the clients need to be protected against privacy attacks. Furthermore, the identity of a particular client will never need to be verified, as much as the ability of that client to send data to the server.

4. Insider attack

The privacy concerns expressed so far were described by considering the possibility that an adversary

might be listening in on the communications channel. The data traffic, however, does not have to be monitored only at the communications channel; if an insider (for example a disgruntled worker) gains access to the central server, then he might be able to access not only the data that travels through the network, but also the data that is stored on the server. Therefore, the server should at no time be able to violate the privacy requirement of the system, even if an adversary has complete control over it and can not only listen to the communications channel, but also manipulate the data that is originating from the server. This latter method of attack will be discussed more under Active Attacks.

As far as securing the devices themselves, our only concern is that any required software does not disclose information in an inappropriate manner, where the appropriateness of disclosure is fully determined by our privacy and security requirements. We do not need to consider the possibility of the device itself being compromised, because if it is, then the user's privacy is automatically compromised irrespective of our system. Any attacks beyond this scope would fall under the Active Attack category.

2.2.4 Active attacks

The attacks described so far only involved a passive adversary – someone listening in to the communications channel, or secretly monitoring the communications from within the server. Other methods of attacking the system involve an adversary communicating with the server or a client; these kinds of attacks are categorized as 'active attacks,' and are detailed in this section.

1. Server impersonation / Rogue access points

As we already discussed, an attacker can impersonate a server either by setting up a fake server (rogue access point), or by completely taking over an existing one. The implication of this possibility is that any users that are connected to the compromised server are immediately in a vulnerable position. The compromised server is not only listening in very carefully to their traffic, but might also send unexpected and strange requests. Since the legitimate server does need to ask the devices for information, this attack is a possibility. Therefore, even under these conditions, the server must not be able to trace any of the preference lists back to any user.

2. Man in the middle attack

A man in the middle attack would be one step down from server impersonation, since the adversary does not have complete access to all the data the server does. However, the possibilities for attack, and precautions against those attacks, are fairly identical to server impersonation. If the server itself cannot violate the privacy of the clients from the data it listens for and/or receives directly, then anyone staging a man in the middle attack should not be able to violate the privacy of the clients either. The direct implication of this attack is that the network used can never be considered to be secure, and any communication needs to be encrypted for maximum privacy.

3. User impersonation

The final attack we need to consider is when someone pretending to be a user tries to connect to the server and manipulate the data stream in a way that would damage the integrity of the system – for example, by skewing the preferences provided to the server in an unwanted way, in order to push the playlist to genres of music (or specific songs) that should not be played in that setting. To avoid this, the server would need to have the ability to authorize users to join the system, be able to verify authorized users, and ensure that the server only receives data from authorized users.

3 Implementation of the system

3.1 Server

SAMPLED consists of a central server and multiple remote clients. As a result, the two need to communicate wirelessly. The best wireless technology to implement SAMPLED over is Wi-Fi (instead of Bluetooth, TDMA, CDMA, UMTS, GSM, etc). The purpose of using Wi-Fi instead telecommunication protocols, is that Wi-Fi (like Bluetooth) operates in unlicensed bands. Furthermore, with Wi-Fi it is easier to set up an ad-hoc network using existing libraries, thus making Wi-Fi our technology of choice. In practice, any protocol that allows for ad-hoc networks can be used to implement this system.

We have established that the server should never be able to violate the user's trust model. To achieve this, the first thing that needs to be maintained is anonymity. Since the data that reaches the server is necessarily anonymous, the task of preventing the server from tracking the users will have to fall on the device itself.

What the server does need to do, however, is be able to authorize users, verify that a certain user is authorized to send data to the server, and only accept data from those authorized users. The best way to accomplish this task is for the server to generate a number of unique IDs that are valid for only one night, and release them so they can be supplied to users. As a result, each night a user visits that same bar, they will need to obtain a new ID. As a consequence, the server will need to verify that the ID a client provides to authenticate with the system was actually generated by the server itself. As a result, the server will need to have a private encryption key (D_S), which will be used to digitally sign the IDs for later verification ([13] section 6.4).

We also considered having the server blind the IDs so that the server could not trace which user ultimately received each unique ID ([9] section 15.8.5, [13] sections 5.3 and 23.12). However, as we will demonstrate later, such a step would be unnecessary. Each ID will be used several times during the night to verify the client's registration, but the client will never directly send any personal information to the server. Therefore, the only accomplishment of blinding the IDs would be to prevent an adversary from knowing when the patron leaves the bar, an implication that does not pose a privacy violation since physical presence can easily be observed without monitoring the IDs that are being used.

The list of signed IDs is generated in advance and kept by the owner/operator of SAMPLED. The operator does not need to be a trusted intermediary, because if the IDs are modified in any way by the operator SAMPLED will simply be rendered inoperative. The IDs, both unique and signed by the server, are issued on a per-request basis. Therefore, any attempts to modify the IDs would just render them invalid.

When a patron enters the bar with his wireless device, he is physically given an ID that is used to authenticate with the server. Since the server should be able to verify each registered user at a later time, the ID will persist for the duration of the night. Therefore, the ID needs to be treated as a personal password that needs to be kept secret. To accomplish this, the data containing any private IDs that are sent to the server by the devices need to be encrypted. At this point, the best method that can be used to fulfill this requirement is public key cryptography. By using the RSA algorithm [12], the server will generate a set of keys unique to that server. This is required, as SAMPLED will be running at various locations and each of the servers running SAMPLED will need to have its own private key, just in case one of them is compromised. Then, the public key is given to the users with their ID as they enter the bar (as seen in Figure 3).

So far, the server can maintain a list of IDs that are in use on that current night (or session) and is in a position to request any device to verify itself. Alternatively, instead of constantly trying to verify each device that sends data to the server by asking for the ID, all data that is sent to the server can contain that ID and be encrypted using the server's public key. This prevents anyone with access to the communications channel from retrieving personal information, as well as preventing clients who are not authorized from sending information to the server. Since all client communication with the server will contain a unique and

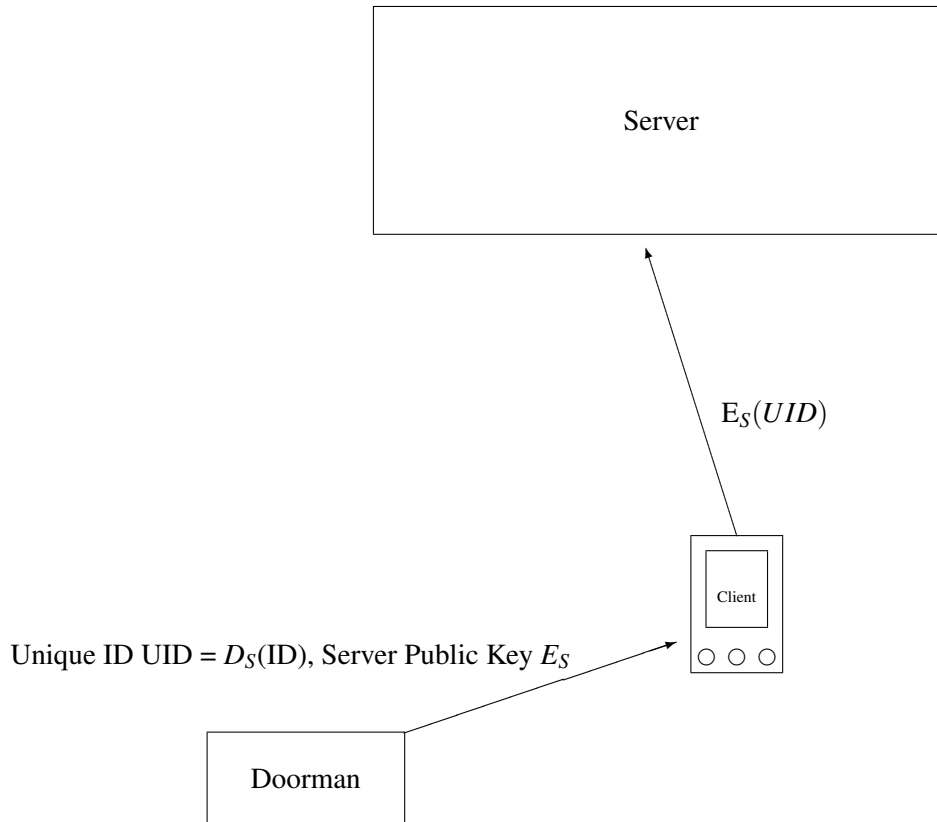


Figure 3: The initial exchange of information when a client first enters the bar

traceable ID, the task of anonymizing the data sent to the server will have to fall on the devices themselves.

Since the server can securely receive data only from registered users, it can now broadcast requests for user preferences from the devices, and receive the feedback it needs to modify the bar's playlist to better match the styles and tastes of the patrons. One more precautionary step needs to be taken here, however, to help the devices from detecting bad queries from compromised servers. Since the devices need to make sure that the data they send as a response to the server's queries cannot be used to violate the user's trust model, some protocol should exist that determines not only what queries the server can send to the devices, but also how often these queries can be sent. The answer to these questions will be determined by the client implementation, since it depends on how the clients choose to deal with the privacy issues.

3.2 Client

As we have already established, the patrons of the bar are the users of the system, and their wireless-enabled devices are the clients. The devices, having the necessary software installed, contain the information the server needs to better adjust the playlist.

As the users enter the bar, they are given the server's public key and a signed ID, which they enter in their device. From this point on, the device will continue running by itself, without any intervention from the user. The first step the client needs to take is to register the ID it was provided with the server, so that the server can mark that id as active. The client can take the signed ID, encrypt it using the server's public key, and send it to the server with an 'activate' request. The server receives the encrypted packet, decrypts

it to retrieve the signed ID, verifies the signature, and marks it as active.

At this point, the server can know which ID corresponds to which client in its network. For this reason when a client receives a query from the server, it cannot directly reply with its response because it will violate the trust model. Therefore, the clients can send the data to the server using a method called Onion Routing[5, 11]. All the registered clients can broadcast their presence to the other clients using zero-configuration networking (or ZeroConf)[2] such as Bonjour[1]. Alternatively, since the IDs that are provided by the server can be associated with individual users without breaking the privacy model, the server can keep a list of the IPs and public keys of the registered clients, which can then be provided on a per-request basis to any client registered with the system. Then, for each client to determine which devices are nearby, the clients can start pinging random IPs from that list and mark the ones that respond as available (see the ‘Distributing public keys’ section for more information). After each client has established who is near them, they can send their data to the server by redirecting the packets they need to send via some number of random nearby clients. The originating client plans the route for the packets in the following manner:

1. The data that needs to reach the server (P) is encrypted using the server’s public key ($M_S = E_S(P)$).
2. The encrypted data (M_S) is placed in a packet, and the packet is addressed to the server ($S.M_S$, where S is the recipient – in this case, the server).
3. A registered client in the network is chosen at random (C_x).
4. The addressed packet ($S.M_S$) is encrypted using that user’s public key ($M_x = E_{C_x}(S.M_S)$).
5. This message is placed in a new packet, and addressed to that user ($C_x.M_x$).
6. Steps 3-5 are repeated a minimum of 5 times so the path is sufficiently randomized, and a packet as in Figure 4 is created.
7. The completed packet is then sent to the last chosen client, C_1 .

When a registered client receives an onion packet addressed to them, they open it, decrypt the packet it contains, and forward the packet to its next destination. Since each layer is encrypted with a different public key, each intermediate client cannot peek at the data before sending it off. As a result, no client can know for certain who the initial sender of the packet they are holding is, nor know the path that same packet was marked with beyond the three hops the client was directly involved with (sender, themselves and recipient).

Figure 5 details an example path that such an onion packet can take, showing 5 hops before the data reaches the server. Each client takes the packet that was sent to them, opens it, decrypts the contents, and forwards the packet to its next destination. Whenever that next destination is the server (in this figure the last hop), the client sending the packet takes their ID, attaches it to the already encrypted message, and encrypts it again using the server’s public key. This way, the server will always know that the packet came from a registered user, but not necessarily the registered user that owns the data being sent.

If not enough registered users are present in the bar to satisfy a minimum requirement (in order to prevent linkability based on the minimum number of hops required for onion routing), then the clients will simply not respond to any queries from the server. This approach cannot be considered to violate the system’s availability, since if not enough clients are around for the minimum number of hops to be effective, then any data collected from that small number of patrons would have little to no effect on the playlist of the bar, as there would simply not be enough information for the server to reach any meaningful state.

To address the question of the frequency of the queries from the server, we need to consider how much data the server needs. Patrons are mobile, some leaving the bar and others entering at all times, so the server needs to maintain updated lists of the preferences. However, the music is being played in the

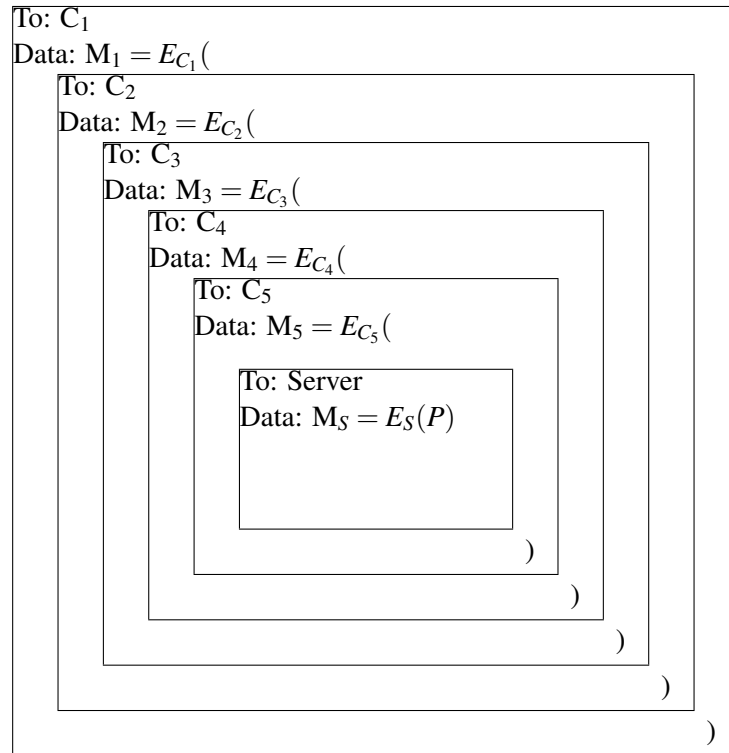


Figure 4: Diagram of onion routing packet, as first constructed by original sender

bar constantly, and the playlist that determines the upcoming songs will need to be determined based on those preferences. With an average song length of 3 minutes, and having the upcoming song always fixed in position, the server should easily handle data that is at least 5 minutes old, if not older, with relative accuracy.

Finally, clients will want to ensure that no data correlation over a number of locations and/or period of time can be used to trace their preferences and their movements. If each time the device is queried for its preferences it provides a complete list, then this correlation would be fairly easy to perform; usually enough songs are present on each preference list to render each list relatively unique. To avoid this violation of the user's privacy model, instead of sending its entire list of preferences every time, the device sends a smaller random subset. It can also choose not to respond to a random number of queries from the server, which will reduce the number of times the client sends its data to the server, thus reducing the probability of any correlation being made. Alternatively, each client can choose a small subset of its preferences at the beginning of the night and use that subset as its complete list of preferences for the duration of that session. This method contains the risk of linkability simply by observing when people leave the bar, and looking at what times certain unique subsets of preferences stopped appearing in the server response list. The problem can be mitigated by randomly choosing which queries to respond to, and which queries to remain silent to. Both these methods sacrifice accuracy of data (and better results on the server side) for privacy.

3.3 Potential privacy and security holes

As with any system, the actual implementation introduces risks that were not accounted for when deciding the mental model. SAMPLED is no different. There are several things we need to consider.

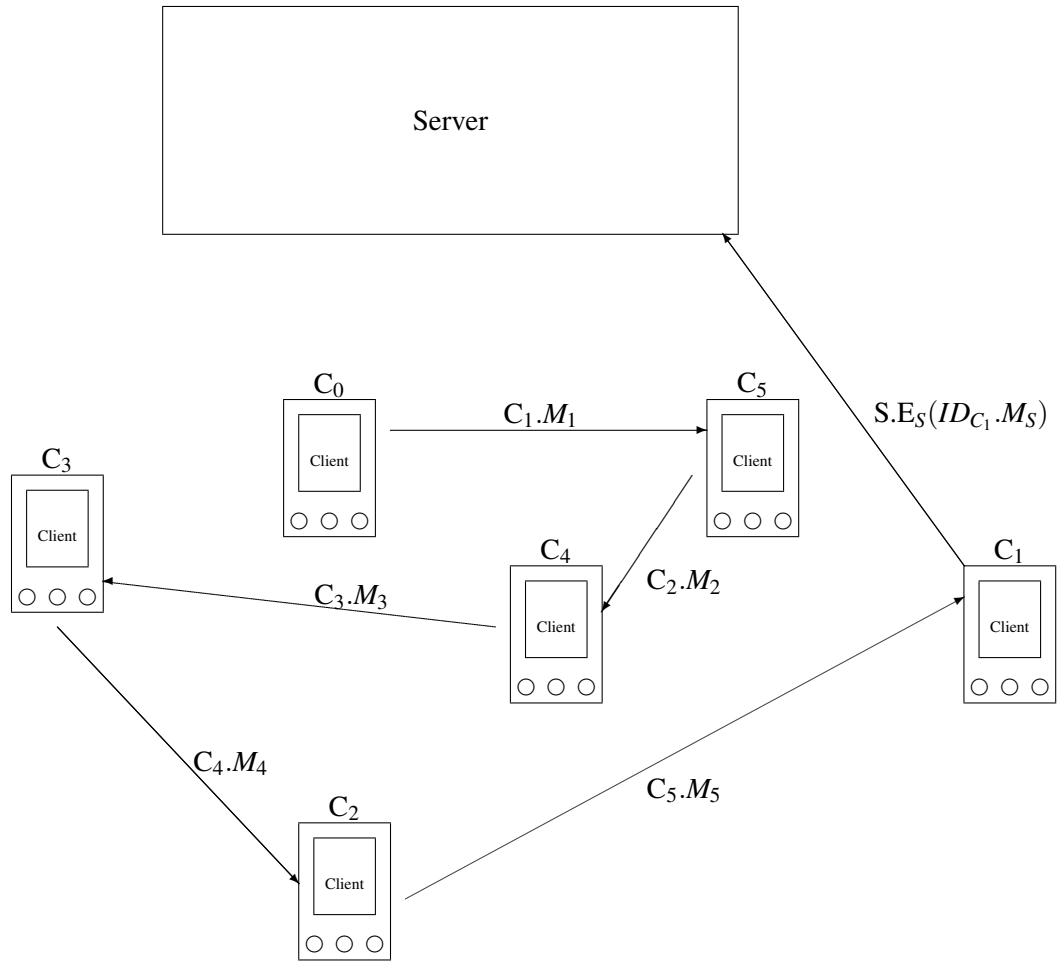


Figure 5: Random route of single onion packet, showing 5 hops

3.3.1 Distributing client public keys

Onion routing requires all communication between clients to be encrypted. For this reason, the clients need to be aware of the public keys of the devices that are near them.

We proposed two ways for distributing the public keys: via a ZeroConf network, or by retrieving a list of authorized users from the server. Both methods have advantages and disadvantages, which are discussed here.

- Retrieving a list via the server

There are several advantages for retrieving the list from the server. Firstly, it is very easy to implement. The existing communications system can be used, since all interested parties can already communicate using encrypted data. Secondly, the server can control who has access to this information. As a result, no unauthorized client can ever be in a position to send any data to the network, as any authorized communication requires knowledge of the public keys of the parties involved, which is very hard to obtain.

One of the disadvantages of this method is that the server cannot know which clients are in a position to directly communicate with each other (since physical proximity plays a very important role when

dealing with the low-power wireless cards that are usually found on mobile devices with limited power supplies). To avoid this problem, the devices can establish who is near them by randomly sending PING commands to clients on that list, and marking who is in a position to reply to them, which introduces potentially unnecessary traffic to an already complicated communication protocol.

The biggest disadvantage of this method, however, becomes apparent if we assume a compromised server with a relatively large number of compromised devices in the same area. Then, when a valid user requests a list of authenticated users, the compromised server responds with a list containing only the compromised clients and the requesting client itself. As a result, any data that is received by the server will immediately be traceable to that single user, since the data was never once bounced through a benign client.

This attack obviously requires the adversary not only to have control of the server (which can include having a rogue access point or staging a man in the middle attack), but also to be in a position to deploy a relatively large number of compromised clients throughout the bar without being noticed. If the adversary, however, is Alice, who already controls both the bar and the server, then she can easily deploy a number of compromised clients around her bar which will mount the attack. In this case, there is almost nothing that can be done to ensure the user's privacy.

- ZeroConf network

Distributing the public keys using a ZeroConf network has clear advantages with respect to the problems faced by retrieving the list via the server. Firstly, the process of determining who is close by becomes much simpler. Each client simply broadcasts their presence and listens for broadcasts from either client. A compromised server cannot control this.

However, a similar problem still exists. Nothing prevents compromised clients that are not part of the authorized network to broadcast their presence to other authorized clients, and therefore make themselves part of the onion routing network. Additionally, nothing prevents those same clients from sending onion packets to the server containing false data, with the intention of injecting the playlist with unwanted music. The server verifies the integrity of each message it receives by examining the ID that the last client attaches to the packet. If a compromised client chooses an authorized client as the last hop of the onion route, then the server will happily accept the packet and add the data to its database.

Additionally, with enough compromised clients becoming part of the network, the probability increases that at least one of the hops of each onion route created by the authorized clients will involve a compromised client. Then, if all the compromised clients simply kill every onion packet they receive instead of forwarding it, that data will never reach the server. Assume there are n neighboring clients in the system, where k of those clients are evil ($k \leq n$), and each client will choose exactly 5 random clients from the complete list of n clients to form its onion route. If $k \geq (1 - (1 - \epsilon)^{1/5})n$ where $0 < \epsilon < 1$, then, on average, more than ϵ -fraction of all queries will result in zero data reaching the server. This shows that if $\epsilon = 0.1$, then $k \approx 0.02n$, which means if more than two percent of the clients are evil, then the server will receive zero responses from the clients 10% of the time, which is a significant percentage for a very small number of compromised devices. This attack is essentially a DoS attack, as it renders the entire system unusable.

In addition, if we again assume a compromised server with a large number of compromised devices in the same area, a similar problem occurs. The probability that the user's privacy will be compromised is no longer 100% since the client will select its route from all the devices that are nearby, not only the compromised ones; however, if the vast majority of the nearby devices is compromised, then this probability becomes quite high.

A last disadvantage of ZeroConf network is the complexity of implementing it. Since the software will need to be deployed on mobile devices, not enough free and standard libraries are available for those devices that will implement ZeroConf.

We believe that the ideal solution would involve a ZeroConf network for distributing the public keys. The main problem with this approach is that adversaries could inject the server with unwanted data. Several features of SAMPLED, however, give us a way of avoiding this attack. Each session of SAMPLED lasts an entire night. Furthermore, SAMPLED can generate a random private/public key pair which it uses for secure communication with the clients and, lastly, already requires clients to manually receive a private unique ID each and every night. Therefore, as a precautionary measure, we can force the server to change its private/public key pair every night, and bundle its public key with all the IDs it hands out on each night. That way, the server's public key can also remain private, and any adversaries that want to enter the onion routing network would still be able to, but would not be in a position to inject data to the server since the server's public key is never broadcast, and changes every night.

Both solutions, however, still remain vulnerable to extreme attacks, where an adversary controls both the server and also has a large number of compromised clients operating in the same bar. The ZeroConf network still has an advantage in this scenario, first because the user has a lower probability of his data being compromised due to the uncompromised source of the neighboring devices, but also because it is possible to completely prevent the attack using methods discussed in section 4.5.

3.3.2 RSA and message length

When data is encrypted using the RSA algorithm, there is still some critical (for this implementation) information that can be determined for the original data. The most critical information in our case is the length of the original data. Since each user is selecting a random subset of their songs to send to the server, it is possible that most clients will select a random subset of the same size (provided they have large enough lists of songs). If this happens, then by observing the size of the encrypted packet that is somewhere in the middle of its onion route, that client can determine how many hops that packet has gone through. As a result a compromised client can gain some information that could be used to trace the original sender of that data. To avoid this attack, several precautions need to be taken.

1. Each client must not only select a random subset of its data, but also a random length of data.
2. Pad the subset with a random number of bytes (to be ignored by the server) so that the size of each packet cannot be used to determine their source.
3. When constructing the onion layers for each hop, pad each layer with a random number of bytes, so that the packet cannot be located between hops by tracing the packet which reduces in size by a determined number of bytes.

3.3.3 Common ultimate recipient

Ultimately, all packets will have to reach the server. To ensure that a client cannot determine any information about the source of the packet when the client receives a packet that needs to be forwarded to the server (for example that the client who sent that packet could not be the original source of the data), a couple of easy precaution steps can be taken. First, when choosing the random route for the onion packet, each client should include itself in the list that is used for the hops. And secondly, the server should also be included as a potential intermediate hop. If the server receives a packet that contains another encrypted packet with a recipient, then the server can very easily forward that packet on, as it will not be able to read its contents.

3.3.4 Traffic analysis

Onion routing is implemented as a synchronized network. All packets are forwarded at the same time (for example on a clock tick) to ensure that no adversary can trace the path of a single packet by simply following each node it is sent to without confusing it with other packets. However, to implement this timing, we would require a secure time server in order to synchronize all wireless devices. Assuming for a second that we have such a capability, we can estimate the security of this implementation (i.e. the probability that a user's privacy might be compromised by employing traffic analysis techniques).

Each client in the network selects 5 random clients from the entire network to form the onion route, and then randomizes the order between those 5 clients. One particular user's privacy might be compromised if the user's route does not intersect with the route of any other user in the system, at any single point. With n clients in the network, then the probability that the privacy of any single user is compromised in such a way is indicated in Equation 1.

$$\lim_{n \rightarrow \infty} \left[\left(\frac{n-1}{n} \right)^n \right]^5 = \frac{1}{e^5} \quad (1)$$

The probability for such a case is very low (less than 1% with 5 hops), and it can be reduced even more simply by increasing the number of hops for the onion route. However, we can also reduce this probability by using heuristics. More specifically, we can introduce a random time interval before each client sends each packet it has received. This way, the probability for an intersection in the path does not only increase, but also the order the packets are received and sent is also randomized, which introduces more variables in our equation.

This approach requires a secure and trusted time server, which we do not have access to. This same system can, however, be implemented in a completely asynchronous manner by using the same heuristics. A unit of time is introduced, which would correspond to one clock period by the time server. Each client then delays forwarding any packet it is currently holding by a random integer x units of time, while continuously listening for incoming packets. If a client that is holding a packet receives a second one, then an intersection in the path of the onion route of two clients has occurred, which protects the system against traffic analysis attacks.

4 Evaluation and testing

Provided with the specifications, precautions, implementation procedures and methods of attack discussed so far, this is how we would implement the SAMPLED system.

4.1 Programming platform

SAMPLED will be deployed on a large number and variety of mobile devices. As a result, the most readily available platform that can be used to develop the required software for the mobile devices has to be Java, and specifically J2ME. Currently, however, there are limitations with J2ME. Most devices that use J2ME (such as PalmOS) implement the limited (CLDC) version, even though the devices themselves have enough processing power to implement a larger set of the language. This subset of Java does not yet have support for any ZeroConf protocol, and limited support for encryption. These features could be implemented if a lower-level language was used (such as C), but this would involve a more tedious and more specialized development of the software, and would be difficult to support a wide range of devices.

4.2 Onion routing

An implementation of onion routing already exists in the form of the Tor network [5]. However, the Tor network is an implementation for bi-directional communication, which is not required by SAMPLED. Furthermore, the Tor network requires a trusted proxy server, which is not available in the SAMPLED network (for more information see section 5). In fact, the only aspects of onion routing that are required were already explained and can be implemented on the J2ME platform. Even without a ZeroConf protocol library available, the public keys can be supplied by the server, as already mentioned.

4.3 Music discovery

Finally, we discuss the part of the server that will process the data and create the playlist. We already mentioned the Audioscrobbler application by Last.fm, which collects data from each users, sends the data to the Last.fm servers, which then process it to provide recommendations to that user. A SAMPLED server could potentially behave as a single user in the Last.fm network and maybe create a contract with Last.fm that will enable the data from each SAMPLED server to be prioritized, with results provided back in real time.

Alternatively, an application such as MuseBox can be developed. “MuseBox is an multichannel, multioutput network digital audio (mp3) player that uses Music Selector Agents (Muses) to automatically play music [...] from a database of mp3s”[10]. MuseBox is developed under Java, and its implementation allows it to receive text commands via a command line, or over a network. Therefore it is not necessary for the SAMPLED server to manage the playlist directly. Instead, an application like MuseBox can be constructed to receive a list of preferences as input and, using algorithms similar to Last.fm, construct a running playlist from the music the bar has available. The SAMPLED server can then simply send the collected list of preferences to the music server, whose sole job is to manage the playlist.

4.4 Use case

Using this information, we can construct a more accurate description for the operation of the clients and the server, detailing all the steps and functions they will need to perform, as well as any data in addition to the preferences they will need to maintain.

- **Clients**

- Receive an ID and a server public key on startup (ideally via a digital process, i.e. a memory card that already contains the data, but can also be fed in manually on startup).
- Create a packet including the provided ID and the client’s public key, encrypt the packet with the provided public key, and send a registration request to the server (server can have a static IP address).
- Listen for queries from the server.
- Listen for pings from other clients.
- Listen for incoming onion packets.
- Send a request to the server every 5 minutes for a list of registered devices, and process the response, storing the list locally. The request must contain the client’s unique ID for verification, and be encrypted using the server’s public key.
- Upon receipt of an onion packet, decrypt its contents to reveal the next recipient and send it on. If the recipient is the server, append the unique ID to the encrypted data, encrypt again using the server’s public key, and send it.

- Compile a random subset of the preferences stored on them.
- Upon a query request from the server, ping random devices from the saved list, until at least 5 nearby devices are found. If not enough devices respond, do not proceed.
- Construct an onion packet containing the random subset of the preferences (as described), using the 5 randomly selected devices as recipients, in random order.
- Send the onion packet to the last chosen recipient.

- **Server**

- Generate a random private/public key pair for each session.
- Generate a list of unique IDs, each of which needs to be signed by the private key.
- Maintain a list of IDs that can be used for registration.
- Maintain a list of registered users for that night, including each user’s unique ID, their IP address, their public key and a timestamp.
- Listen for incoming data packets from clients.
- Upon receipt of a packet, decrypt it and extract the ID. If the signature on the ID is valid, and the ID is on the list of unused IDs, then move the ID to the list of registered users (which persists for an entire session) and associate it with a timestamp and the client’s public key. If the ID is neither in the list of unused IDs nor in the list of registered users, ignore the packet. Otherwise, process its contents.
- Receive a data packet containing another onion packet, and forward it on.
- Receive a data packet containing the encrypted preferences of a user, decrypt that packet, and add the data to the list of collected preferences.
- Receive a data packet containing a request for the list of available registered users from a specific client, update that client’s timestamp entry in the registered users list to reflect the current time, and generate another list containing the public key and IP of all users whose timestamp is less than 15 minutes old. Then encrypt that list using the requesting client’s public key and send it to the client who sent the request.
- Right before the next query is sent, send the aggregate list of collected preferences to the music server, and purge the list.
- Every 5 minutes send a query request to all clients in the list of registered users whose timestamp is less than 15 minutes old.

4.5 Problems faced

It is clear from the analysis in section 3 that it would be nearly impossible (in the current state of the system’s implementation) for the privacy of the user to be preserved, if the bar in which SAMPLED is deployed is completely overrun by adversaries (i.e. the server as well as a large number of clients are compromised). If Alice wishes to know exactly what kind of music one particular patron, Bob, enjoys listening to, then she can set up an attack to target Bob’s device by using compromised devices that are allowed to be part of the network by the server. If Bob’s device happens to select only compromised devices to send its preferences to the server, then Bob’s data can be recovered very easily.

This kind of attack, however, could be avoided if Bob’s presence was completely anonymized in the network, and a ZeroConf network was used for determining the neighboring clients. To achieve complete anonymity, the unique IDs provided would need to be blinded as described in section 3. Then, in order for

Bob's device to remain anonymous, it will need to employ random MAC addressing [7], which involves Bob's device randomly changing its MAC address every time it needs to send an onion packet, as well as every time it needs to provide the server with Bob's unique ID. Furthermore, instead of using IP addresses to send data from one device to the next (which would involve a potential DHCP server that would be a privacy implication), the actual MAC addresses are used to send the packets.

Finally, as explained in section 3.3.1 under the explanation for the ZeroConf network, the SAMPLED system would be vulnerable to DoS attacks by a relatively small number of unauthorized devices. The reason why this is possible is because when the authorized devices search for other nearby devices, they have no way of determining if the devices broadcasting their presence are registered. Any attempts to verify each client would lead to potential privacy implications. Therefore, this availability problem exists as a trade-off for privacy. It becomes extremely hard to protect the entire system and at the same time maintain anonymity.

It is possible, however, to detect that a DoS attack of this type is occurring. Each device that enters the bar is given a private and unique ID. To register that ID with the system, the client communicates with the server directly, so the presence of compromised clients killing the onion routing will not affect the registration of clients. As a result, if the server continues to log registration requests by clients yet never receives any data, it can assume that the entire system is under a DoS attack, and raise an alert.

5 Related work

SAMPLED takes advantage a lot of existing technologies that have already been mentioned, such as Last.fm's music recommendation system and the new wave of social music sharing coming with systems such as Push!Music. There are several more systems available, however, that operate on the same principle. One big example is Pandora, found at www.pandora.com. By providing Pandora just one of your favorite artists, it can provide you with a personalized station that contains not only music by that artist and other popular artists that create similar music, but by many other artists that might be obscure, and were recommended by other users as being the same type of music as the favorite artist you provided. In general, music discovery tools are becoming increasingly popular, increasingly accurate, and even more abundant [3]

Another system that is based on user preferences to make music recommendations is the iTunes MiniStore. The iTunes MiniStore is also a great example of how concerned users are with their privacy. When Apple released the 6.0.2 version of their iTunes application, the version contained a little advertised feature called the iTunes MiniStore. This little feature would monitor what songs were being played by iTunes and send information to some servers in order to retrieve related products from the iTunes music store. There was no visible way to turn it off, and Apple did not say how the data was collected, nor what they were doing with the data. Internet users were most annoyed by the fact that they weren't given a warning and a choice rather than the privacy violation itself. Apple had established a trust model with its users, and falsely assumed that the MiniStore would not gather any privacy concerns. In the wake of the Sony root-kit publicity, where Sony, a large and presumably respectable record label, bundled a small program with some of their audio CDs, which silently installed itself on the user's computer as soon as the audio CD was inserted and stealthily sent data of that user's actions to Sony, people became much more aware of privacy and security problems forced on them by large corporations. Very quickly, it became obvious that not only the privacy implications, but Apple's attitude towards those privacy implications, played a large role in the user's response to this feature. [4, 6]

The main component of SAMPLED that secures most of its anonymity is onion routing. Many researchers have looked at onion routing, and a large scale application is currently deployed in the form of the Tor network. [5, 11] The Tor network is deployed to provide anonymity for popular actions on the public Internet, such as browsing, email, and even file transfer. As already mentioned, this network is implemented

for 2-way communication, with each node in the onion route retaining information about who sent what packet, so that any responses can be sent back to where they originated. More importantly, the Tor network is implemented using a trusted third party, called a proxy, which is used to coordinate the communication as well as the timing of the onion packets, which need to be synchronized. Given the speed at which ordinary users are used to browsing the Internet, a secure time server comes in handy as randomizing heuristics cannot be implemented without imposing a severe delay on all communication over the Tor network. The SAMPLED onion network, on the other hand, is entirely one-way, with the clients all sending information to the central server. Additionally, since the clients have a window of several minutes between queries from the server, certain delays can be introduced to the system that will not affect the ultimate timely collection of data.

6 Conclusion and future directions

We believe that SAMPLED is a realistic future application that could receive widespread use. We have demonstrated a real problem concerning privacy and security and have found no existing solutions that directly address this problem. Other people have looked at this area of data exchange, as well as the attendant privacy issue, but have done little to combine the two. SAMPLED was based on existing technologies and explored different ways of bringing these technologies together to create a system that would satisfy our requirements.

Even though our proposed system cannot yet be implemented due to the current state of the technology, it is our belief that the continuing trends of related technologies will enable the creation of this system within the near future. SAMPLED depends on the patrons of the bar owning and using a wireless-enabled music playing device that contains at least a representative sample of their musical preferences – a dependency which is not currently in widespread occurrence.

One observation from the proposed system is that privacy and security can rarely be provided together in a complete form. It is often the case, as with the DoS attack described in section 3.3.1, that some security must be sacrificed to achieve a higher level of privacy, or vice versa. As a consequence, we acknowledge that we have not addressed every possible problem posed by the implementation of this system, yet we expect that future solutions will address them.

Some potential solutions were proposed, though not explored, in this paper. One solution for completely anonymizing each and every user in the system was proposed, which would potentially protect the privacy of the users even from an entirely compromised system. The proposed solution involved blinding the IDs that were handed out, and also using random MAC addressing to hide the true identity of each user from the entire system. The potential implications of this approach with respect to the other potential methods of attack were not explored.

Additionally, we did not explore the possibility of using digital cash ([13] section 6.4), which would be almost equivalent to using unique, signed, blinded IDs for authentication.

Looking away from the future and past the technical problems of the present, we could also explore the possibilities for implementing this system with what is available right now. A lot of mobile devices that can support protocols such as Bluetooth and WiFi – for example Blackberries and PDAs – support J2ME and are already popular amongst many groups of people. Since systems, such as Pandora, that can provide recommendations based on a single artist name already exist, it is not hard to envision a system where users (not devices) are polled for their preferences by asking them if they enjoy a certain song, or a certain artist. That data might just prove to be enough so that the system can modify the default playlist in a way that would satisfy more of its listeners.

Another key element of this paper was the importance users place on the privacy of their listening habits. Anecdotes such as the iTunes MiniStore suggest that concern does exist about this exact form of

privacy. Yet systems like Last.fm exist, which provide raw, unfiltered information of users' listening habits to the Internet. The main difference between these two cases is that users of Last.fm enroll in the system having full knowledge of what kind of personal information will be made available to the public, and they opt-in.

This paper has not examined the psychological factors that entertain users' opinions about privacy. It is possible that the proposed trust model does not match the mental trust model. Our model is that the clients can never trust the server. If, however, we are a regular in a bar that uses our proposed system, that might not be true. For example, how many users would trust the server if it offered WPA encryption? Or how many users would trust the entire system if, instead of getting its information directly from the user's device, small touch screen panels were scattered around the entire area of the bar, which users could then use to indicate their preference for one song, or their dislike for another?

These are all questions that we expect will be answered in the near future, and maybe provide an improved solution to our proposed problem.

References

- [1] I. Apple Computer. Networking - Bonjour. <http://developer.apple.com/networking/bonjour/index.html>, 2005. Last visited on May 28, 2006.
- [2] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of IPv4 link-local addresses. Technical report, RFC 3927, 2005.
- [3] C. Dahlen. Better than we know ourselves. <http://www.pitchforkmedia.com/features/weekly/06-05-22-better-than-we-know-ourselves.shtml>, May 2006. Last visited on June 1.
- [4] C. Doctorow. iTunes update spies on your listening and sends it to Apple? http://www.boingboing.net/2006/01/11/itunes_update_spies_.html, January 2006. Last visited on May 31.
- [5] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.
- [6] R. Griffiths. Eyeing the iTunes MiniStore. <http://www.macworld.com/weblogs/editors/2006/01/ministore/index.php>, January 2006. Last visited on May 31.
- [7] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: A quantitative analysis. *ACM Mobile Networks and Applications (MONET)*, 10:315–325, 2005.
- [8] M. Håkansson, M. Jacobsson, and M. Rost. Push!Music. <http://www.viktoria.se/fal/projects/music/index.html>, Autumn 2006. Last visited on May 20, 2006.
- [9] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice-Hall, 2nd edition, 2002.
- [10] C. Michael. MuseBox distributed player. <http://cmichae.acm.jhu.edu/musebox/home.htm>, 2004. Last visited on May 31.
- [11] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Protocols using anonymous connections: Mobile applications. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols Workshop*, pages 13–23, 1997.
- [12] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [13] B. Schneier. *Applied Cryptography*. John Wiley and Sons, Inc., New York, 2nd edition, 1996.
- [14] Wikipedia. Social network. http://en.wikipedia.org/wiki/Social_network. Last visited on May 20, 2006.